GESTURE VOICE CONNECT

A Project Report Submitted to "JNTU-GV, Vizianagaram" For Fulfilment of Requirements for Award of Degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

TARUN KUMAR POTTI **20NR1A0587**

Under the guidance of

Dr. S. Vidya Sagar Appaji

Associate Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BABA INSTITUTE OF TECHNOLOGY AND SCIENCES

(Approved by A.I.C.T.E & Affiliated to J.N.T.U.G.V, Vizayanagaram)

PM Palem, Visakhapatnam District, Andhra Pradesh

(2020-2024)

BABA INSTITUTE OF TECHNOLOGY AND SCIENCES

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project work entitled entitled **"GESTURE VOICE CONNECT"** is the bonafide work submitted by **P. Tarun Kumar (20NR1A0587)** in partial fulfilment of the requirements for the award of degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE ENGINEERING** from **BABA INSTITUTE OF TECHNOLOGY & SCIENCES** (Affiliated to JNTUGV, Vizianagaram).

Project Guide	**Head of the Department**
Dr. S. Vidya Sagar Appaji	Mrs. U. Padma Mohan
Associate Professor	Associate Professor

DECLARATION

We hereby declare that this report for a project entitled **“GESTURE VOICE CONNECT”** has been developed by us, under the supervision of **DR. S. VIDYA SAGAR APPAJI**, Associate Professor in Department of **COMPUTER SCIENCE & ENGINEERING**, **BABA INSTITUTE OF TECHNOLOGY AND SCIENCES** during the academic year 2020-2024 and has not been submitted to any other university for the award of any kind of degree.

Tarun Kumar Potti **(20NR1A0587)**

ACKNOWLEDGMENT

We would like to take this opportunity to express my deepest gratitude to my project supervisor, **DR. S. VIDYA SAGAR APPAJI, Associate Professor, COMPUTER SCIENCE & ENGINEERING, BABA INSTITUTE OF TECHNOLOGY & SCIENCES**, P.M. Palem, Visakhapatnam, who has persistently and determinedly guided me during the whole course of this project.

We are grateful to **Mrs. U. PADMA MOHAN, Head of the Department, Department of Computer Science and Engineering**, for providing us with the required facilities for the completion of the project work.

We are very much thankful to the Principal **Dr. B POORNA SATYANARAYANA and Management of BITS**, for their encouragement and cooperation in carrying out this work.

We express our thanks to all teaching faculty of the **Department of CSE**, whose suggestions during reviews helped us in the accomplishment of our project. We would like to thank all nonteaching staff of the **Department of CSE** for providing great assistance in the accomplishment of our project.

We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. At last, but not the least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

Tarun Kumar Potti **(20NR1A0587)**

ABSTRACT

In an era characterized by the rapid evolution of human-computer interaction, the convergence of hand gesture recognition and voice assistance emerges as a promising frontier for intuitive and seamless communication between humans and machines. At the heart of this endeavor lies the ambition to create a robust system that seamlessly integrates hand gesture recognition with voice assistance, harnessing the capabilities of cutting-edge computer vision and speech recognition technologies.

The project endeavors to develop a sophisticated system capable of accurately interpreting hand gestures in conjunction with voice commands. By leveraging the power of computer vision, the system aims to detect and recognize various hand movements in real-time, facilitating intuitive interactions with machines. Simultaneously, through the integration of advanced speech recognition algorithms, the system will enable users to articulate commands and queries effortlessly using natural language.

This fusion of technologies represents a significant step forward in enhancing user experience and accessibility across a wide array of applications and domains. By bridging the gap between physical gestures and verbal commands, the system offers users a more natural and intuitive means of interacting with computers and devices. Whether controlling home appliances, navigating digital interfaces, or accessing information, the seamless coordination of hand gestures and voice commands promises to revolutionize the way we engage with technology.

Through the synergy of computer vision and speech recognition technologies, this project seeks to unlock new possibilities for human-computer interaction, paving the way for a future where communication with machines is more intuitive, efficient, and enriching than ever before.

INDEX

Contents

1. INTRODUCTION

1.1 Feasibility study -

Technical Feasibility

The project leverages well-established technologies and libraries, indicating a high degree of technical feasibility. The use of Python as the programming language, along with OpenCV and MediaPipe for computer vision, and Speech Recognition for voice recognition, provides a solid foundation for development.

OpenCV and MediaPipe are widely used and well-documented libraries for computer vision tasks, including hand gesture recognition. Their robust hand tracking and landmark detection capabilities make them suitable choices for this project.
Similarly, Speech Recognition libraries are readily available in Python, enabling the integration of voice assistance functionality. These libraries have been widely used and tested, increasing the likelihood of successful implementation.

However, the seamless integration of hand gesture recognition and voice assistance may require additional research and development efforts to ensure smooth coordination between the two modalities.

Operational Feasibility

The proposed system aims to provide an intuitive and seamless user experience by combining hand gestures and voice commands. This approach has the potential to enhance accessibility and user-friendliness, making it operationally feasible across various domains.

In healthcare settings, the hands-free operation of medical equipment can improve sterility and efficiency, addressing a practical need. Similarly, in educational environments, the system can provide alternative means of interaction for individuals with disabilities, enhancing inclusivity.

The application of the system in smart home automation aligns with the growing trend of voice-controlled devices and home automation systems, further increasing its operational feasibility.

Economic Feasibility

The project primarily relies on open-source libraries and tools, which can significantly reduce development costs. Python, OpenCV, MediaPipe, and Speech Recognition libraries are freely available, minimizing the need for expensive proprietary software or licenses.

However, the development effort required for seamless integration, testing, and deployment may incur costs related to human resources and computing infrastructure. Additionally, if the project aims to commercialize the system, there may be additional costs associated with marketing, sales, and support.

Overall Feasibility

Based on the information provided in the abstract, the "Gesture Voice Assist" project appears to be feasible from a technical standpoint, given the availability of well-established libraries and technologies. The proposed applications demonstrate operational feasibility, addressing practical needs across various domains.

The economic feasibility largely depends on the project's scope, development effort, and potential commercialization plans. While the use of open-source tools can reduce costs, additional expenses related to human resources, infrastructure, and potential commercialization efforts should be considered.

To further assess feasibility, it would be beneficial to conduct a more detailed analysis of the specific technical challenges, resource requirements, and potential market demand for such a system.

1.2 Existing System -

Hand Gesture Recognition System

Video Input: The hand gesture recognition system relies on live video input, likely from a camera or webcam, to capture the user's hand movements and gestures.

Computer Vision Libraries: The project utilizes well-established computer vision libraries like OpenCV and MediaPipe to process the video input and detect hand gestures.

Hand Tracking: Specifically, the abstract mentions the use of MediaPipe's hand tracking module, which employs advanced algorithms and machine learning techniques to accurately detect and track hand landmarks in real-time.

Gesture Interpretation: By analysing the positions and movements of the tracked hand landmarks, the system can interpret and recognize various hand gestures. This could involve predefined gestures, such as closed fists, open palms, or specific finger movements.

Voice Assistance System

Speech Recognition: The voice assistance component of the system incorporates speech recognition technology to interpret spoken commands from users.

Speech Recognition Libraries: The abstract mentions the use of libraries like Speech Recognition, which likely leverage machine learning models trained on large speech datasets to accurately transcribe and understand spoken utterances.

Command Interpretation: Once the spoken commands are transcribed, the system needs to interpret their meaning and determine the appropriate actions or responses based on the recognized gestures and voice input.

Response Generation: Depending on the interpreted command and recognized gesture, the voice assistance system can generate relevant responses, which could include executing specific tasks, providing information, or controlling connected devices or applications.

Integration and Coordination

The key innovation proposed in the "Gesture Voice Assist" project is the seamless integration and coordination between the hand gesture recognition and voice assistance systems. The abstract suggests that specific hand gestures will serve as triggers to activate the voice assistance mode, allowing users to interact with the system through a combination of gestures and spoken commands.

For example, a predefined gesture like a closed fist or an open palm could prompt the system to start listening for voice commands. Once a voice command is recognized, the system would interpret the command in the context of the previously detected gesture and execute the appropriate action or response.

This integration aims to provide a more natural and intuitive way of interacting with computers and devices, leveraging the strengths of both hand gesture recognition and voice assistance technologies.

1.3 Proposed system -

Multimodal Interaction

The proposed system aims to enable multimodal interaction by fusing hand gesture recognition and voice assistance capabilities. This allows users to interact with computers or devices using a combination of hand gestures and voice commands.

Gesture-Voice Coordination:

- The system is designed to seamlessly coordinate hand gestures and voice commands.
- Specific hand gestures, such as a closed fist or an open palm, serve as triggers to activate the voice assistance mode.
- Once activated, the system listens for voice commands and executes corresponding tasks based on the recognized gesture and voice input.

Core Functionality:

- Real-time hand gesture recognition from live video input.
- Accurate interpretation of various hand movements and gestures.
- Voice command recognition and interpretation.
- Seamless coordination between recognized gestures and voice commands.
- Execution of appropriate tasks or responses based on the multimodal input.

The proposed system aims to revolutionize human-computer interaction by offering an innovative and intuitive way to control computers and devices through the fusion of hand gestures and voice commands. It leverages cutting-edge technologies in computer vision and speech recognition to provide enhanced user experience and accessibility across diverse applications.

1.3.1 Proposed system Advantages -

Intuitive and Natural Interaction: The fusion of hand gesture recognition and voice assistance provides a more intuitive and natural way for humans to interact with computers and devices. It leverages the natural human abilities of hand movements and speech, making the interaction more seamless and user-friendly.

Enhanced Accessibility: The multimodal interaction approach can greatly benefit individuals with disabilities, offering alternative means of interaction beyond traditional input methods. For example, users with limited mobility can rely on voice commands, while those with speech impairments can utilize hand gestures

Hands-Free Operation: The ability to control systems through voice commands and hand gestures enables hands-free operation, which is particularly useful in scenarios where maintaining sterility is crucial, such as in healthcare settings.

Improved User Experience: The seamless coordination between hand gestures and voice commands can lead to a more engaging and immersive user experience, as users can interact with the system in a more natural and intuitive manner.

Versatility: The proposed system has a wide range of potential applications across diverse domains, including healthcare, education, and smart home automation, as highlighted in the abstract.

Leveraging Advanced Technologies: The system utilizes cutting-edge technologies in computer vision (OpenCV, MediaPipe) and speech recognition, benefiting from the latest advancements in these fields.

Open-Source Tools: The abstract mentions the use of open-source libraries like OpenCV and Speech Recognition, potentially reducing development costs and promoting collaboration and community contributions.

Multimodal Input: By combining hand gestures and voice commands, the system can leverage the strengths of both input modalities, potentially leading to more robust and efficient interaction.

Overall, the "Gesture Voice Assist" system aims to provide a more accessible, intuitive, and seamless human-computer interaction experience by integrating advanced technologies in computer vision and speech recognition, while offering versatility and potential cost savings through the use of open-source tools.

2. SOFTWARE REQUIREMENTS SPECIFICATIONS

2.1 Introduction -

2.1.1 Purpose

The purpose of this is to define the software requirements for the Gesture Voice Assist system. This system aims to provide a robust and intuitive human-computer interaction solution by integrating hand gesture recognition and voice assistance capabilities.

2.1.2 System Overview

The Gesture Voice Assist system will leverage computer vision and speech recognition technologies to enable users to interact with computers and devices through a combination of hand gestures and voice commands. The system will utilize Python programming language, along with OpenCV and Medi-Pipe computer vision libraries for real-time hand gesture detection, and speech recognition libraries for voice command interpretation.

The core functionality of the system will allow users to activate voice assistance by performing predefined hand gestures, such as a closed fist or an open palm. This seamless coordination between hand movements and voice commands will provide an efficient and accessible interface for various applications, including healthcare, education, and smart home automation.

2.1.3 Scope

This software requirement specification will cover the following key aspects of the Gesture Voice Assist system;

Hand Gesture Recognition Module

- Capturing live video input.
- Detecting and tracking hand landmarks using Media-Pipe.
- Recognizing predefined hand gestures.

Voice Assistance Module

- Integrating speech recognition capabilities.
- Interpreting voice commands and executing corresponding actions.

Gesture-Voice Interaction

- Triggering voice assistance mode through hand gestures.
- Coordinating hand gestures and voice commands for seamless user interaction.

System Integration and Applications

- Integrating the hand gesture recognition and voice assistance modules.
- Exploring potential use cases and applications, such as healthcare, education, and smart home automation.

2.2 External Interface Requirements -

User Interface

The system shall feature an intuitive graphical user interface (GUI) that provides visual feedback to the user regarding hand gesture recognition and voice command interpretation. It shall include clear instructions and visual cues to guide users in performing hand gestures and issuing voice commands. The GUI should be adaptable to different screen sizes and resolutions to ensure compatibility across various devices.

Hardware Interface

The system shall be compatible with a wide range of hardware configurations, including laptops, desktop computers, and mobile devices equipped with cameras and microphones. It shall support both built-in and external cameras for capturing live video input, with provisions for adjusting camera settings such as resolution and frame rate.

The system should be optimized to minimize hardware resource requirements while maintaining optimal performance during real-time hand gesture recognition and speech processing tasks.

Software Interface

The system shall interface seamlessly with computer vision libraries such as OpenCV and Media Pipe to access hand tracking and gesture recognition functionalities. It shall integrate with speech recognition libraries like Speech Recognition to enable accurate interpretation of voice commands.

The software interfaces should adhere to industry standards and best practices to facilitate interoperability with third-party applications and services.

Communication Interface

The system shall establish communication channels between the hand tracking module and the speech recognition module to synchronize hand gestures with voice commands. It shall support data exchange protocols for transmitting real-time hand gesture data and voice input between the various system components. The communication interface should prioritize low latency and high throughput to ensure responsive interaction between the user and the system.

2.3 Functional Requirements -

Hand Gesture Recognition Module:

- The system shall capture live video input from a camera.
- The system shall detect and track hand landmarks using the MediaPipe hand tracking module.
- The system shall recognize predefined hand gestures based on the detected hand landmarks.
- The system shall be able to distinguish between different hand gestures, such as a closed fist, an open palm, a pointing finger, and others.
- The system shall provide real-time hand gesture recognition with low latency.
- The system shall be able to handle variations in hand size, orientation, and lighting conditions.

Voice Assistance Module:

- The system shall capture audio input from a microphone.
- The system shall use speech recognition libraries to interpret the user's voice commands.
- The system shall be able to recognize a predefined set of voice commands.
- The system shall provide appropriate responses or execute corresponding actions based on the recognized voice commands.

- The system shall have the capability to provide audio feedback or responses to the user.

Gesture-Voice Interaction:

- The system shall allow users to activate the voice assistance mode by performing a predefined hand gesture.
- The system shall be able to coordinate the recognized hand gestures and voice commands to enable seamless user interaction.
- The system shall be able to handle situations where both hand gestures and voice commands are used in combination to execute a task.
- The system shall provide clear visual and/or audio feedback to the user to indicate the active mode (hand gesture recognition or voice assistance).

System Integration and Applications:

- The system shall be able to integrate the hand gesture recognition and voice assistance modules into a cohesive solution.
- The system shall be designed to be modular and extensible, allowing for easy integration with various applications and use cases.
- The system shall be able to support use cases in healthcare, education, and smart home automation, among others.
- The system shall provide configuration options to customize the recognized hand gestures and voice commands for specific application requirements.
- The system shall be able to handle multiple users interacting with the system simultaneously.

These functional requirements outline the key capabilities and features that the Gesture Voice Assist system should possess to enable seamless human-computer interaction through the fusion of hand gestures and voice commands.

2.4 Non-Functional Requirements –

Performance

The system shall provide real-time hand gesture recognition with a maximum latency of 100 milliseconds. The system shall be able to process voice commands with a response time of less than 500 milliseconds. The system shall be able to maintain a stable frame rate of at least 30 frames per second during hand gesture recognition.

Reliability

The system shall have an accuracy rate of at least 90% in recognizing predefined hand gestures. The system shall have a voice command recognition accuracy rate of at least 85%. The system shall be able to operate without critical failures for at least 99.9% of the time.

Usability

The system shall provide intuitive and user-friendly interfaces for hand gesture and voice command interactions. The system shall offer clear and informative visual and audio feedback to the user during interactions. The system shall be easy to set up and configure for various application requirements. The system shall have comprehensive documentation and tutorials to support user onboarding and training.

Scalability

The system shall be able to handle multiple simultaneous users without significant performance degradation. The system shall be designed to accommodate future expansion and integration with additional hardware or software components. The system shall be able to scale its processing capabilities to handle increased computational demands as the user base or application complexity grows.

Portability

The system shall be compatible with multiple operating systems, including Windows, macOS, and Linux. The system shall be designed to be platform-independent, allowing for deployment across different hardware configurations. The system shall have minimal dependencies on specific hardware or software components, enabling easy deployment and maintenance.

Maintainability

The system shall have a modular and well-documented architecture to facilitate easy maintenance and updates. The system shall provide mechanisms for remote diagnostics and troubleshooting to support efficient issue resolution.

3. ANALYSIS

Introduction

UML stands for Unified Modelling Language. UML is a standardized general-purpose modelling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components. The Primary goals in the design of the UML are as follows:

Provides users a ready-to-use, expressive visual modelling Language so that they can develop and exchange meaningful models.

- Provide extensibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development process.
- Provide a formal basis for understanding the modelling language.
- Encourage the growth of OO tools market.
- Support higher level development concepts such as collaborations, frameworks.
- patterns and components.
- Integrate best practices.

Building Blocks of the UML

The vocabulary of the UML encompasses three kinds of building blocks:

- Things
- Relationships
- Diagrams

Things are the abstractions that are first-class citizens in a model; relationships tie these things together; diagrams group interesting collections of things.

Things in the UML

There are four kinds of things in the UML

- Structural things
- Behavioural things
- Grouping things
- Annotational things

Relationships

It illustrates the meaningful connections between things. It shows the association between the entities and defines the functionality of an application. There are four types of relationships given below:

- Dependency
- Generalization
- Realization
- Association

Diagrams

The diagrams are the graphical implementation of the models that incorporate symbols and text. Each symbol has a different meaning in the context of the UML diagram. There are thirteen different types of UML diagrams that are available in UML 2.0, such that each diagram has its own set of a symbol. And each diagram manifests a different dimension, perspective, and view of the system diagrams are classified into three categories that are given below;

- Structural Diagram
- Behavioural Diagram

3.1 Use Case Diagram

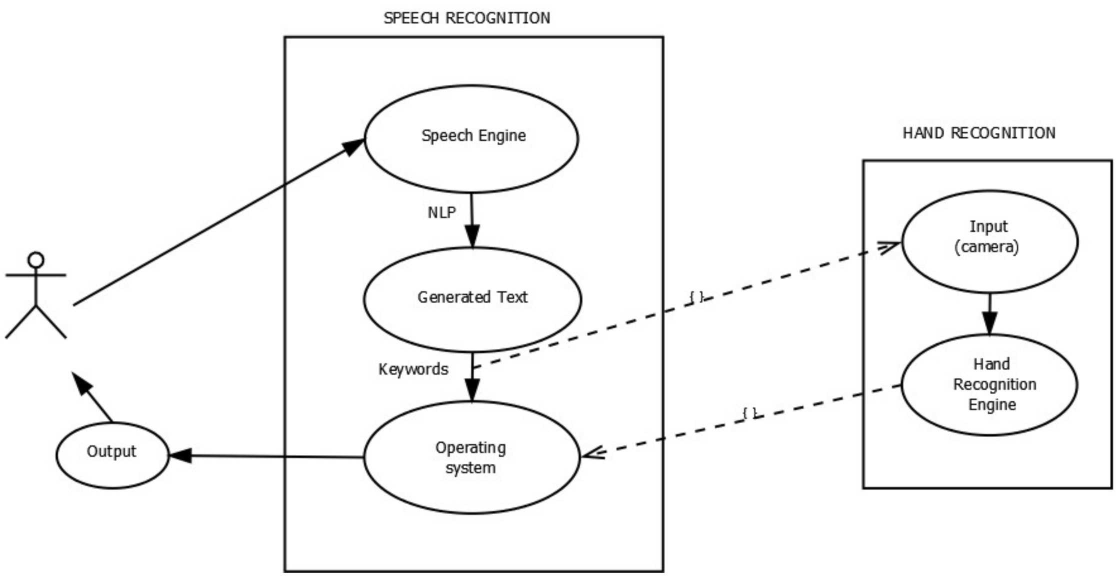


Fig 3.1: Use Case Diagram

Actor: User

Description: The primary actor interacts with the system through hand gestures and voice commands to perform tasks seamlessly.

Use Cases

Capture Input: The system captures live video input from a camera and audio input from a microphone.

Detect Hand Gestures: Utilizes computer vision libraries to process live video input and detect hand gestures accurately.

Recognize Speech: Integrates a speech recognition module to interpret spoken commands received via the microphone.

Coordinate Gestures and Voice: Coordinates between hand gestures and voice commands, activating voice assistance based on specific hand actions.

Execute Tasks: Executes tasks based on recognized gestures and spoken commands, interfacing with various functionalities or applications.

Relationships

- The User initiates actions such as Capturing Input, Detecting Hand Gestures, and Recognizing Speech.

- The system responds by Coordinating Gestures and Voice, Executing Tasks, and Providing Feedback to the User.

3.2 Sequence Diagram

The sequence diagram represents the flow of messages in the system and is also termed as an event diagram. It helps in envisioning several dynamic scenarios. It portrays the communication between any two lifelines as a time-ordered sequence of events, such that these lifelines took part at the run time.

In UML, the lifeline is represented by a vertical bar, whereas the message flow is represented by a vertical dotted line that extends across the bottom of the page. It incorporates the iterations as well as branching.

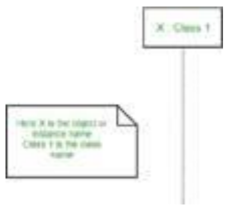
Purpose of a Sequence Diagram

- To model high-level interaction among active objects within a system.
- To model interaction among objects inside a collaboration realizing a use case.
- It either models' generic interactions or some certain instances of interaction.

SEQUENCE DIAGRAM NOTATION

1. Actors: An actor in a UML diagram represents a type of role where it interacts with the system and itsobjects. It is important to note here that an actor is always outside the scope of the system we aim to model using the UML diagram.

2. Lifelines: A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically, each instance in a sequence diagram is represented by a lifeline. Lifeline elementsare located at the top in a sequence diagram. The standard in UML for naming a lifeline follows the following format – Instance Name: Class Name.


X : Class 1
Here X is the object or instance name Class 1 is the class name

3. Messages: Communication between objects is depicted using messages. The messages appear in a sequential order on the lifeline. We represent messages using arrows. Lifelines and messages form the core of a sequence diagram.

4. Guards: To model conditions we use guards in UML. They are used when we need to restrict the flow of messages on the pretext of a condition being met. Guards play an important role in letting software developers know the constraints attached to a system or a particular process. For example: In order to be able to withdraw cash, having a balance greater than zero is a condition that must be met asshown below.

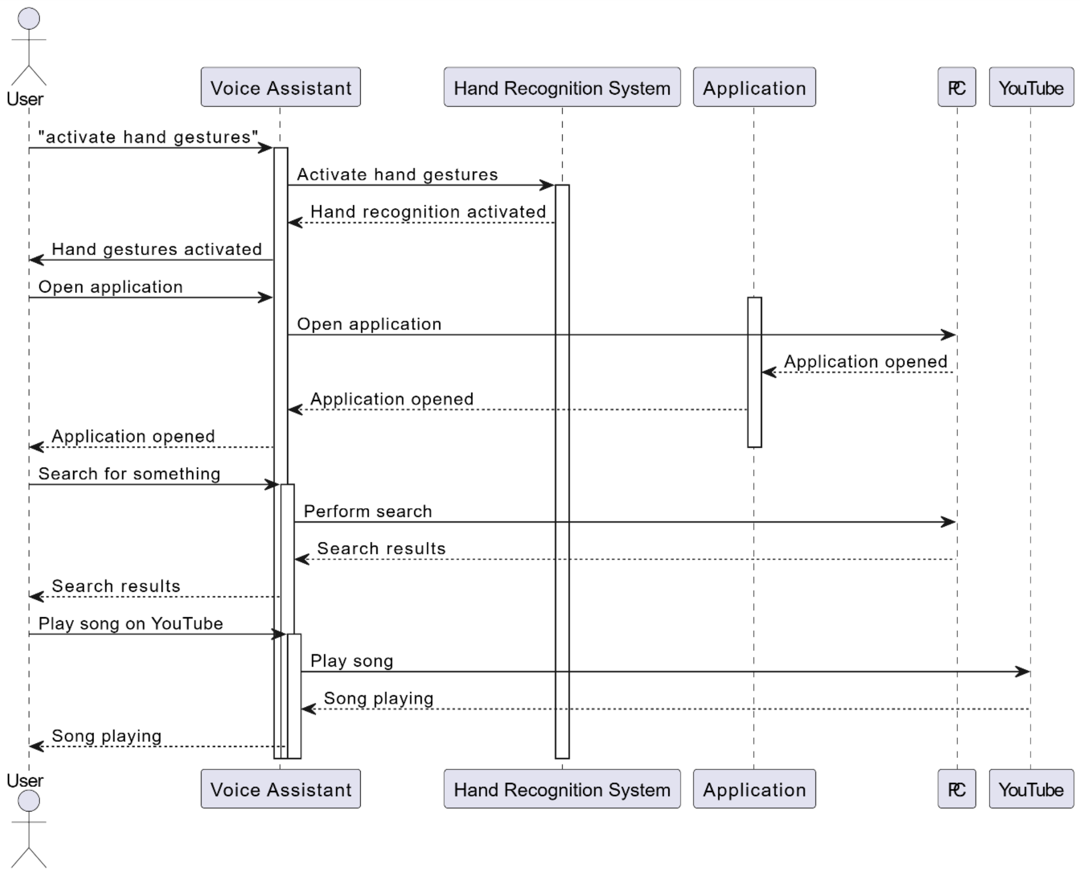


Fig 3.2: Sequence Diagram

The sequence diagram illustrates the flow of interactions within the Hand Gesture Recognition and Voice Assistance system. Initially, the Input Module captures live video feed from a camera and audio input from a microphone. The Hand Gesture Recognition Module processes the video input using OpenCV and Media Pipe to detect hand gestures, while the Speech Recognition Module interprets spoken commands from the microphone input.

Upon detecting specific hand gestures, the Gesture-Voice Coordination Module triggers the system to listen for voice commands. When a voice command is recognized, the Task Execution Module executes corresponding tasks based on the recognized gestures and spoken commands. These tasks may involve interfacing with various applications or functionalities. Finally, the User Interface Module provides feedback to the user through visual and/or auditory means, enhancing the overall user experience.

The Integration and Deployment module ensures seamless integration of all components using Python as the primary programming language, facilitating deployment on various platforms.

3.3 Collaboration Diagram

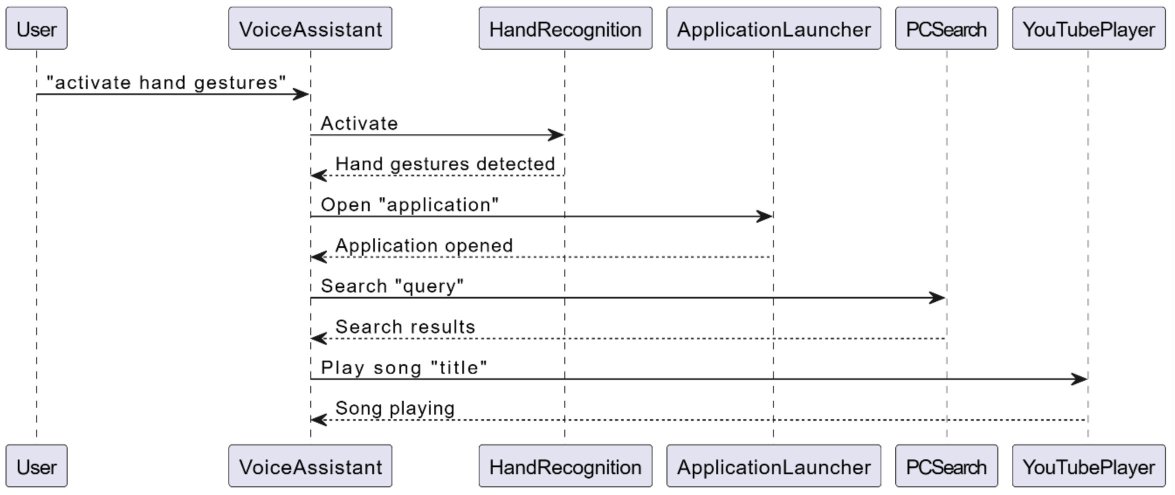


Fig 3.3: Collaboration Diagram

- The collaboration diagram illustrates the interaction between various modules within the Hand Gesture Recognition and Voice Assistance System. It depicts how each component collaborates to enable seamless communication between humans and machines.

- The "Input Module" captures input from the camera and microphone, providing data to subsequent modules. The "Hand Gesture Recognition Module" processes live video input using OpenCV and Media Pipe, detecting hand gestures in real-time. Simultaneously, the "Speech Recognition Module" interprets audio input from the microphone using libraries like Speech Recognition.
- These modules feed their outputs into the "Gesture-Voice Coordination Module," which coordinates between recognized hand gestures and spoken commands. Once a trigger gesture activates voice assistance, the "Task Execution Module" executes corresponding tasks based on recognized gestures and commands. Feedback to the user is provided through the "User Interface Module."
- Finally, the entire system is integrated and deployed using Python as the primary language, ensuring seamless operation across various applications and domains. This collaboration ensures efficient and intuitive human-computer interaction, enhancing user experience and accessibility.

3.4 State chart diagram

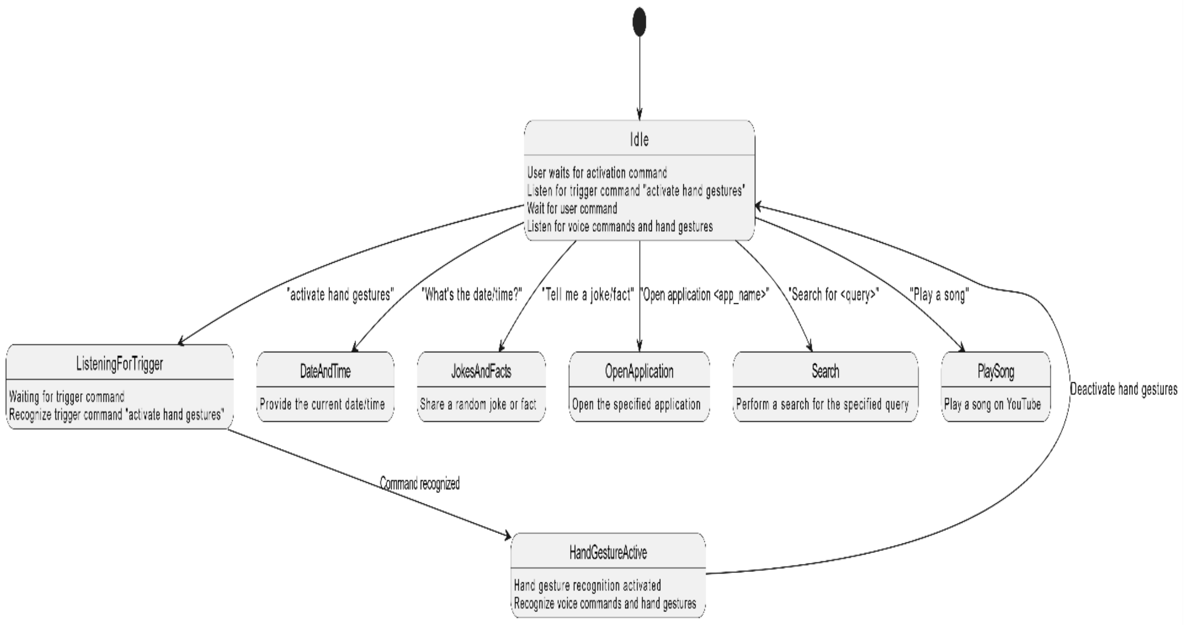

Idle
User waits for activation command
Listen for trigger command "activate hand gestures"
Wait for user command
Listen for voice commands and hand gestures
"activate hand gestures"
"What's the date/time?"
"Tell me a joke/fact"
"Open application <app_name>"
"Search for <query>"
"Play a song"
ListeningForTrigger
Waiting for trigger command
Recognize trigger command "activate hand gestures"
DateAndTime
Provide the current date/time
JokesAndFacts
Share a random joke or fact
OpenApplication
Open the specified application
Search
Perform a search for the specified query
PlaySong
Play a song on YouTube
Deactivate hand gestures
Command recognized
HandGestureActive
Hand gesture recognition activated
Recognize voice commands and hand gestures

Fig 3.4: State Chart Diagram

- The state chart diagram represents the various states and transitions within the proposed hand gesture recognition and voice assistance system. Initially, the system is in an idle state, awaiting input from the user. Upon detecting a hand gesture, it transitions to a state where it analyses the gesture using computer vision techniques, facilitated by OpenCV and Media Pipe libraries.

- If a gesture is recognized, the system transitions to a state where it listens for voice commands using the Speech Recognition module. Upon receiving a valid voice command, the system transitions to a state where it executes the corresponding task, such as controlling devices or triggering actions in applications. Throughout this process, the system maintains a state of coordination between hand gestures and voice commands, ensuring seamless interaction.
- After task execution, the system returns to the idle state, ready to receive new inputs. This state chart diagram encapsulates the dynamic behaviour of the system, illustrating how it interprets user gestures and voice commands to facilitate intuitive communication and interaction.

3.5 Activity diagram

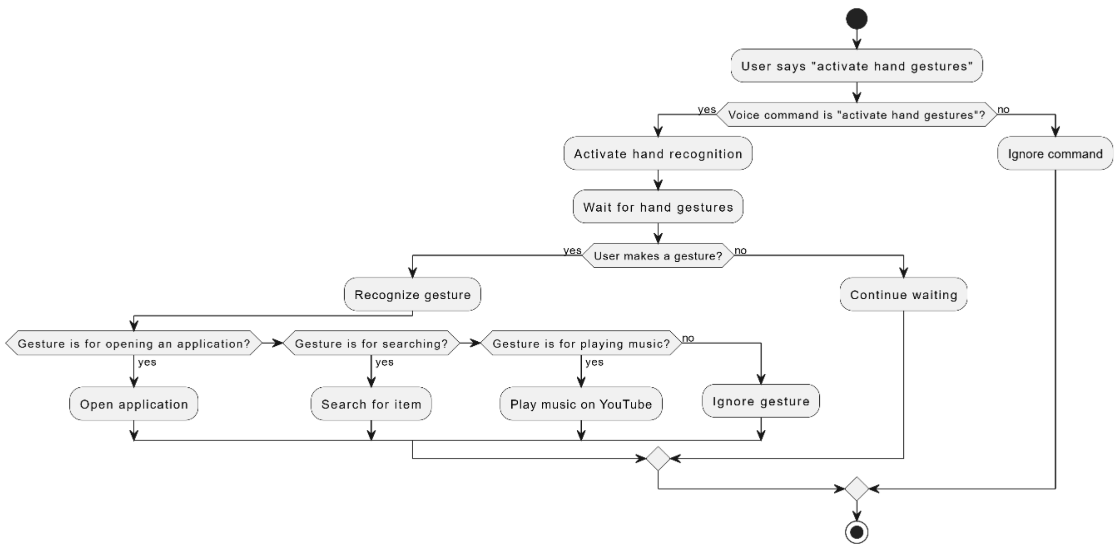

User says "activate hand gestures"
yes
Voice command is "activate hand gestures"?
no
Activate hand recognition
Ignore command
Wait for hand gestures
yes
User makes a gesture?
no
Recognize gesture
Continue waiting
Gesture is for opening an application?
Gesture is for searching?
Gesture is for playing music?
no
yes
yes
yes
Open application
Search for item
Play music on YouTube
Ignore gesture

Fig 3.5: Activity Diagram

The activity diagram illustrates the flow of actions within the Hand Gesture Recognition and Voice Assistance System described in the abstract. The diagram begins with the Input Module, where data is captured from the camera and microphone. This data is then processed by the Hand Gesture Recognition Module, which utilizes computer vision libraries like OpenCV and Media Pipe to detect hand gestures in real-time. Simultaneously, the Speech Recognition Module interprets spoken commands received from the microphone.

Following this, the Gesture-Voice Coordination Module synchronizes the hand gestures detected with the spoken commands recognized. If a specific gesture triggers the system to listen for voice commands, the Task Execution Module executes tasks based on the combined inputs of recognized gestures and spoken commands. These tasks are diverse and application-specific, ranging from controlling medical equipment in healthcare settings to managing home appliances in smart homes.

This activity diagram encapsulates the dynamic interaction between different components of the system, emphasizing the seamless coordination between hand gestures and voice commands to execute tasks across various applications and domains.

4. DESIGN

4.1 Architecture

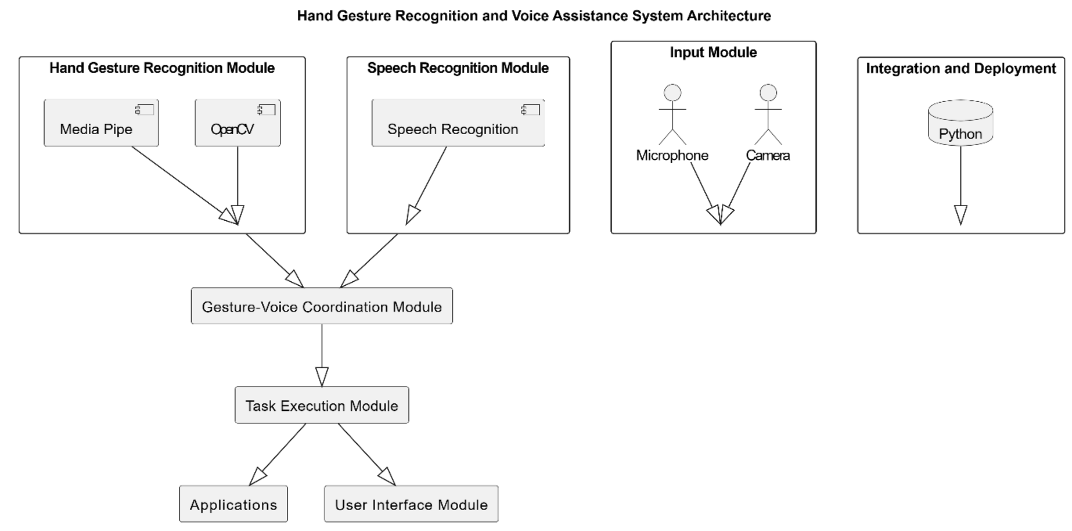

Hand Gesture Recognition and Voice Assistance System Architecture
Hand Gesture Recognition Module
Media Pipe
OpenCV
Speech Recognition Module
Speech Recognition
Input Module
Microphone
Camera
Integration and Deployment
Python
Gesture-Voice Coordination Module
Task Execution Module
Applications
User Interface Module

Fig 4.1: System Architecture

The architecture for the proposed Hand Gesture Recognition and Voice Assistance system integrates various modules to enable intuitive human-computer interaction. At its core, the system encompasses an Input Module, capturing data from live video feed and microphone input. The Hand Gesture Recognition Module utilizes computer vision libraries like OpenCV and Media Pipe for real-time processing, accurately detecting hand movements and gestures. Simultaneously, the Speech Recognition Module employs libraries such as Speech Recognition to interpret spoken commands from the microphone.

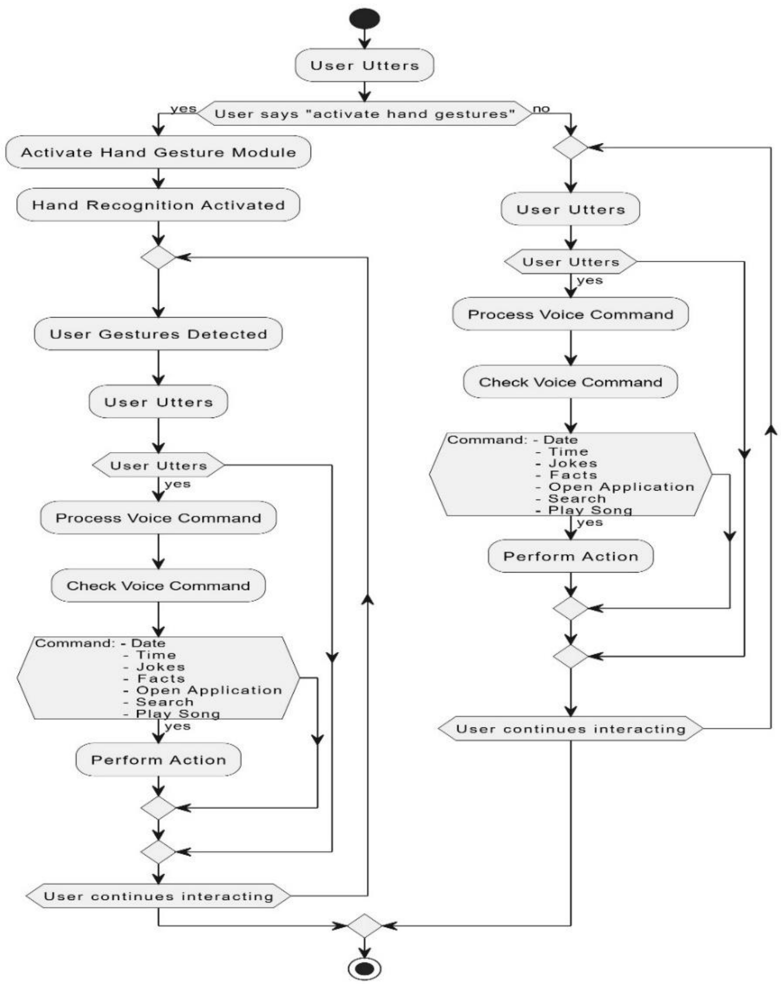
These modules converge in the Gesture-Voice Coordination Module, where specific hand actions serve as triggers for activating voice assistance. For instance, gestures like a closed fist or open palm prompt the system to listen for voice commands. Upon recognition, the Task Execution Module executes tasks corresponding to the detected gestures and voice commands, interfacing with various functionalities or applications.

The system's versatility extends to diverse applications such as healthcare, education, and smart home automation. In healthcare, it facilitates hands-free operation of medical equipment, aiding professionals in sterile environments.

Educational settings benefit from enhanced accessibility for individuals with disabilities, enabling alternative interaction with computers. Moreover, in smart home automation, users can effortlessly control appliances and systems through intuitive gestures and voice commands.

Python serves as the primary programming language, orchestrating the integration and deployment of the system. The User Interface Module ensures seamless feedback to users, enhancing their overall experience. Through the fusion of hand gesture recognition and voice assistance, this architecture aims to revolutionize human-computer interaction, fostering accessibility and usability across various domains.

4.2 Project flow



User Utters
yes
User says "activate hand gestures"
no
Activate Hand Gesture Module
Hand Recognition Activated
User Gestures Detected
User Utters
User Utters
yes
Process Voice Command
Check Voice Command
Command: - Date - Time - Jokes - Facts - Open Application - Search - Play Song
yes
Perform Action
User continues interacting
User Utters
User Utters
yes
Process Voice Command
Check Voice Command
Command: - Date - Time - Jokes - Facts - Open Application - Search - Play Song
yes
Perform Action
User continues interacting

Fig 4.2: Project Flow

The voice project encompasses various functionalities triggered by user commands. Upon uttering "activate hand gestures," the system activates hand recognition, enabling users to interact through gestures. Additionally, users can request information such as the date, time, jokes, and facts. Furthermore, the facilitates opening applications installed on the user's system and conducting searches for any query.

Moreover, it supports playing songs on YouTube in response to commands like "play a song." Through natural language processing and integration with diverse modules, the seamlessly processes user requests, executes actions, and provides relevant responses. This comprehensive functionality empowers users to navigate their digital environment efficiently, whether it's obtaining information, accessing applications, or enjoying entertainment, all initiated through simple voice commands.

The key aspects of the flow are:

1. **User Utterances -** The system listens for the user saying "activate hand gestures" to trigger the hand gesture recognition module.
2. **Activate Hand Gesture Module -** This module is responsible for detecting and recognizing the user's hand gestures.
3. **Process Voice Commands -** Once hand gestures are detected, the system can process various voice commands from the user, such as commands related to date, time, jokes, facts, opening applications, searching, and playing songs.
4. **Check Voice Command -** The system checks and validates the voice command provided by the user.
5. **Perform Action -** If the voice command is valid, the system performs the requested action, such as retrieving and displaying the requested information or executing the requested application/task.
6. **User continues interacting -** The flow indicates that the user can continue interacting with the system by providing additional voice commands and gestures.

Overall, this appears to be a multimodal interaction system that combines hand gesture recognition and voice commands to allow users to control various applications and retrieve information through natural interactions

4.3 Class Diagram

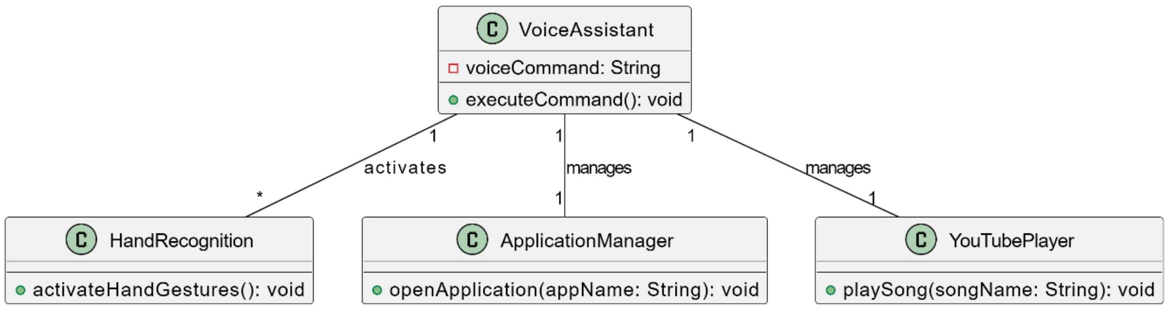


Fig 4.3: Class Diagram

- **Voice (C) -**

Description: The Voice class embodies an intelligent digital designed to interact with users through voice commands. It serves as an intermediary between users and various functionalities within the system, facilitating seamless hands-free operation.

Methods

1. **voiceCommand():** Accepts a string parameter representing the voice command uttered by the user.

2. **executeCommand():** Executes the received voice command by invoking the appropriate functionality or service within the system. This method acts as the core functionality of the voice , enabling it to interpret and respond to user requests effectively.

- **Hand Recognition (C) -**

Description: The Hand Recognition class integrates functionality for recognizing and interpreting hand gestures, enhancing user interaction with the system. By leveraging advanced computer vision techniques, it enables users to control the system through intuitive hand movements.

Methods

activateHandGestures(): Initiates the hand gesture recognition system, enabling users to interact with the system using predefined hand gestures. This functionality augments the overall user experience by providing an alternative input method, particularly useful in scenarios where voice commands may not be feasible or preferred.

- **Application Manager (C) -**

Description: The Application Manager class serves as the system's gatekeeper for managing installed applications and facilitating their execution. It provides a centralized interface through which users can launch and access various applications installed within the system environment.

Methods

openApplication (appName: String) : Opens the specified application identified by its name. This method enables users to seamlessly navigate between different applications, promoting efficient multitasking and accessibility within the system.

- **YouTube Player (C) -**

Description: The YouTube Player class encapsulates the functionality required to play songs or audio content from the YouTube platform. It offers users the ability to stream and enjoy their favourite music directly within the system environment.

Methods

playSong (songName: String) : Initiates the playback of the specified song, allowing users to enjoy audio content from the vast library available on YouTube. This method enhances the entertainment value of the system by integrating seamless access to online multimedia content.

This expanded documentation provides a comprehensive overview of each class's purpose, functionality, and contribution to the overall system architecture. It aims to elucidate the role of each component while highlighting their respective capabilities and benefits to the end user.

4.4 Component Diagram

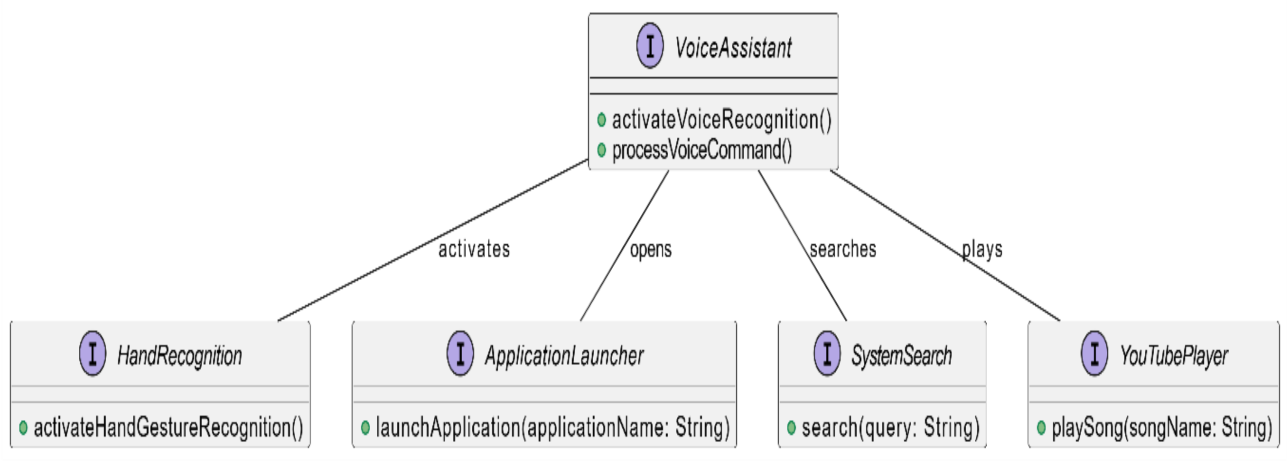


Fig 4.4: Component Diagram

1. Voice -

Description: The Voice class represents an intelligent digital capable of interpreting voice commands and performing various actions within the system.

Methods

activateVoiceRecognition(): Initializes the voice recognition system, enabling the to listen for and interpret voice commands.

processVoiceCommand(): Processes the voice command received by the and executes the corresponding action, which may include activating applications, opening files, searching the web, or playing media content.

Actions supported

- **activates:** Activates specified functionalities or services within the system.
- **opens:** Opens applications or files as per the user's request.
- **searches:** Conducts searches based on the provided query.
- **plays:** Initiates playback of media content such as songs or videos.

2. Hand Recognition -

Description: The Hand Recognition class provides functionality for recognizing and interpreting hand gestures, allowing users to interact with the system using gestures.

Methods

Activate Hand Gesture Recognition(): Initializes the hand gesture recognition system, enabling users to control the system through predefined hand gestures.

3. Application Launcher -

Description: The Application Launcher class facilitates the launching of applications within the system.

Methods

LaunchApplication(applicationName**:** String**):** Launches the specified application identified by its name.

4. System Search -

Description: The System Search class enables users to search for various items or perform actions within the system.

Methods

Search(query: String): Conducts a system-wide search based on the provided query.

5. YouTube Player -

Description: The YouTube Player class provides functionality for playing songs or videos from the YouTube platform.

Methods

playSong(songName**:** String**):** Initiates the playback of the specified song from YouTube.

This documentation outlines the classes and their respective methods, elucidating their roles and functionalities within the system. It provides a comprehensive overview of the system's capabilities, including voice recognition, hand gesture recognition, application launching, system-wide search, and YouTube playback.

5. IMPLEMENTATION

5.1 Introduction to Technology

The implementation of the system leverages the following technologies and tools;

Programming Language: Python

Python is an interpreted, high-level, general-purpose programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991. Python emphasizes code readability and a syntax that allows programmers to express concepts in fewer lines of code compared to other languages, making it particularly suitable for beginners and experienced developers alike.

Features

1. **Simple and Readable Syntax:** Python's syntax is designed to be simple and easy to understand, with a focus on readability. This simplicity reduces the learning curve for new developers and facilitates collaboration among team members.

2. **Dynamic Typing:** Python is dynamically typed, meaning you don't need to specify variable types explicitly. This allows for more flexible and concise code, but it also requires careful attention to variable types during development to avoid runtime errors.

3. **Extensive Standard Library:** Python comes with a comprehensive standard library that provides modules and packages for a wide range of tasks, from file I/O and networking to web development and data processing. This extensive library reduces the need for external dependencies and accelerates development.

4. **Interpreted and Interactive:** Python is an interpreted language, which means that code is executed line by line, making it easier to debug and test. Additionally, Python supports interactive mode, allowing developers to execute code snippets and experiment with features in real-time using the Python interpreter.

5. **Object-Oriented and Functional Programming:** Python supports both object-oriented and functional programming paradigms, allowing developers to

choose the approach that best fits their needs. This flexibility enables the creation of modular, reusable, and maintainable code.

6. **Platform Independent:** Python code is platform-independent, meaning it can run on various operating systems without modification. This cross-platform compatibility makes Python suitable for developing applications that need to run on different environments seamlessly.
7. **Libraries and Frameworks:** Python has a vast ecosystem of third-party libraries and frameworks that extend its functionality for specific tasks and domains. For example, libraries like NumPy and pandas are popular for data analysis and scientific computing, while frameworks like Django and Flask are widely used for web development.
8. **Package Management:** Python's package management system, pip, makes it easy to install, manage, and distribute packages and dependencies. With pip, developers can quickly install libraries from the Python Package Index (PyPI) and keep their project dependencies up-to-date.

Libraries

- **OpenCV (Open Source Computer Vision Library)**

OpenCV stands as an **open-source computer vision** and machine learning library renowned for its extensive toolkit and versatility. Initially developed by Intel and now maintained by an active community, OpenCV serves as a foundational tool for developers delving into image and video processing tasks, among others.

Purpose: OpenCV aims to provide developers with a robust set of tools and algorithms for various computer vision tasks, including but not limited to image processing, object detection, motion tracking, and machine learning integration.

Features

Image Processing: OpenCV offers a wide array of functions for fundamental and advanced image processing operations, empowering developers to perform tasks such as filtering, edge detection, and morphological transformations.

Object Detection and Recognition: With algorithms like Haar cascades, HOG, and deep learning-based models, OpenCV facilitates accurate object detection and recognition in images and videos.

Video Analysis: OpenCV enables comprehensive video analysis, including motion detection, optical flow estimation, and object tracking.

Machine Learning Integration: OpenCV seamlessly integrates with popular machine learning libraries like scikit-learn and TensorFlow, allowing developers to combine computer vision techniques with advanced machine learning algorithms.

Cross-Platform: OpenCV boasts cross-platform compatibility, supporting various operating systems including Windows, Linux, macOS, Android, and iOS.

- **Community and Documentation:** The OpenCV community actively contributes to its development, providing extensive documentation, tutorials, and examples to support developers' learning and implementation efforts.

• Media Pipe

Media Pipe emerges as an innovative open-source framework developed by Google, designed to facilitate the creation of real-time multimedia processing pipelines.

Purpose: Media Pipe aims to streamline the development of multimedia processing applications by offering a flexible framework equipped with pre-built components for tasks like object detection, hand tracking, and pose estimation.

Features

Pre-built Components: Media Pipe provides a comprehensive collection of pre-built components, each encapsulating complex algorithms for various perceptual tasks. These components can be easily customized and integrated into bespoke pipelines.

Flexibility: Developers can leverage Media Pipe's flexibility to craft tailored pipelines that cater to specific use cases, empowering them to combine and configure pre-existing components seamlessly.

Integration with TensorFlow: Media Pipe seamlessly integrates with TensorFlow, enabling the incorporation of machine learning models into multimedia processing pipelines for enhanced functionality and performance.

Scalability: Media Pipe exhibits robust scalability across different hardware platforms, ensuring efficient real-time processing of multimedia data across desktop, mobile, and embedded devices.

- ## Speech Recognition

Speech Recognition emerges as a Python library aimed at simplifying the integration of speech recognition capabilities into applications, offering developers a high-level interface to various speech recognition engines.

Purpose: Speech Recognition serves as a user-friendly solution for incorporating speech recognition functionalities into Python applications, catering to a wide range of use cases.

Features

Multiple Engine Support: Speech Recognition supports multiple speech recognition engines, including Google Web Speech API, Sphinx, Wit.ai, and Microsoft Bing Voice Recognition, providing developers with flexibility in choosing the engine that aligns with their requirements.

Language Support: With support for multiple languages, Speech Recognition accommodates international applications, enabling developers to implement speech recognition in diverse linguistic contexts.

Easy Integration: Speech Recognition boasts a simple and intuitive API, facilitating seamless integration into Python applications without extensive coding overhead.

Customization: Developers can customize recognition parameters and handle various audio formats with ease, tailoring the speech recognition process to specific project needs and optimizing performance and accuracy. These libraries represent essential tools for developers embarking on projects involving computer vision, multimedia processing, and speech recognition, offering a rich array of features and functionalities to address diverse application requirements.

- ## PyAutoGUI: Automating GUI Interaction with Python

PyAutoGUI is a Python library designed to automate graphical user interface (GUI) interactions. It offers a straightforward yet robust set of functions for controlling the mouse and keyboard, capturing screenshots, and interacting with windows and controls.

PyAutoGUI simplifies the process of automating repetitive tasks in GUI-based applications, making it an invaluable tool for developers, testers, and power users alike.

Features

Mouse Control: PyAutoGUI facilitates precise control over mouse movements, allowing developers to navigate the cursor to specific coordinates on the screen. It also empowers users to simulate mouse clicks, double-clicks, and drags, along with scrolling actions, enabling seamless interaction with GUI elements.

Keyboard Input: The library enables programmatically typing text and special characters, facilitating automated data entry tasks. PyAutoGUI supports the simulation of key presses and releases, including modifiers like Shift and Ctrl, and the dispatching of keyboard shortcuts to applications.

Screen Interaction: Developers can leverage PyAutoGUI to capture screenshots of the entire screen or designated regions, providing valuable visual data for analysis and processing. The library also offers image recognition capabilities, enabling the identification and localization of specific visual elements on the screen.

Window Management: PyAutoGUI empowers users to interact with windows and controls, including minimizing, maximizing, and closing windows, as well as activating and switching between applications. It facilitates the automation of tasks involving window manipulation, such as navigating through dialog boxes and interacting with UI components.

• Pyttsx3: Text to Speech

Pyttsx3 is a Python library that serves as a text-to-speech (TTS) conversion tool, allowing developers to integrate speech synthesis capabilities into their Python applications. With Pyttsx3, developers can convert written text into spoken words, providing auditory feedback to users or enabling applications to communicate verbally with users. It utilizes the Text-to-Speech (TTS) engine provided by the underlying platform, making it compatible with various operating systems such as Windows, macOS, and Linux.

Features

Cross-Platform Compatibility: One of the key features of Pyttsx3 is its cross-platform compatibility. It leverages the native text-to-speech engines available on different operating systems, ensuring consistent performance and behaviour across Windows, macOS, and Linux platforms. This makes Pyttsx3 suitable for developing applications that target a wide range of devices and environments.

Simple API: Pyttsx3 provides a simple and intuitive API for performing text-to-speech conversions. Developers can easily incorporate speech synthesis functionality into their Python scripts with minimal effort. The API allows developers to specify the text to be synthesized, adjust speech parameters such as voice and rate, and control the playback of synthesized speech.

Customizable Voices: Pyttsx3 offers support for multiple voices, allowing developers to customize the speech output according to their preferences or application requirements. Users can choose from a variety of voices with different accents, genders, and languages, enabling applications to deliver speech in a manner that suits the context or target audience. Additionally, developers can adjust parameters such as pitch, volume, and speed to further customize the speech output.

Real-Time Synthesis: Pyttsx3 supports real-time synthesis of speech from text, enabling applications to generate spoken feedback or prompts dynamically based on user input or system events. This real-time synthesis capability is essential for interactive applications such as voice assistants, virtual agents, or accessibility tools, where timely auditory feedback enhances user experience and interaction.

Non-Blocking Operation: Pyttsx3 operates in a non-blocking manner, allowing developers to perform text-to-speech conversions asynchronously without blocking the main execution thread. This non-blocking behaviour ensures that speech synthesis operations do not disrupt the responsiveness of the application's user interface or impede the execution of other tasks. Developers can initiate speech synthesis operations in the background and continue with other processing tasks concurrently.

Extensibility: Pyttsx3 is highly extensible, allowing developers to customize and extend its functionality to suit specific requirements. Developers can integrate Pyttsx3 with other Python libraries or frameworks to enhance its capabilities or integrate it into larger software systems. Additionally, Pyttsx3 supports call backs and event handling mechanisms, enabling developers to respond dynamically to speech synthesis events such as speech completion or errors.

- **PyWhatKit**

PyWhatKit is a Python library that provides a simple yet powerful interface for performing various tasks commonly encountered in day-to-day programming. It serves as a handy toolkit for developers, offering functionalities ranging from sending WhatsApp messages to performing Google searches, playing YouTube videos, and much more. Developed by Ankit Raj Mahapatra, PyWhatKit aims to streamline common tasks and simplify automation processes, making it an indispensable tool for Python enthusiasts.

Features

Google Search: The library facilitates Google searches directly from Python, allowing users to retrieve search results programmatically. By simply specifying the search query, PyWhatKit fetches relevant information from the web, making it convenient for extracting data or performing research tasks within Python scripts.

Play YouTube Videos: PyWhatKit offers the ability to play YouTube videos directly from Python scripts. Developers can specify the video URL or search query, and the library will open the corresponding video in the default web browser. This feature is useful for integrating multimedia content into Python applications or scripts.

Send Email: Sending emails programmatically becomes seamless with PyWhatKit. Developers can specify the recipient's email address, subject, and message content, and the library takes care of the rest. This feature is handy for automating email notifications or communication tasks within Python applications.

Text-to-Handwriting: PyWhatKit provides a unique feature to convert text into handwritten notes. By specifying the text content and output file name, developers can generate handwritten notes in PNG format, mimicking the appearance of real handwriting. This feature adds a personalized touch to digital content and is ideal for creating custom notes or memos.

Get Wikipedia Summary: Retrieving summaries from Wikipedia becomes effortless with PyWhatKit. Developers can specify the topic of interest, and the library fetches the corresponding summary from Wikipedia. This feature is useful for accessing concise information on a wide range of topics directly from Python scripts.

Info: PyWhatKit allows users to retrieve information about various entities, including weather forecasts, stock prices, and COVID-19 statistics. By specifying the entity of interest, such as a city name or stock symbol, developers can fetch relevant information from online sources and integrate it into Python applications or scripts.

QR Code Generation: The library provides functionality for generating QR codes from text or URLs. Developers can specify the input text or URL, and PyWhatKit generates a corresponding QR code image. This feature is useful for creating QR codes dynamically and integrating them into digital content or printed materials.

5.2 Sample code

File - HandRecognition.py:

```
'''
the architecture of the  mediapipe module for hand detection be like...

mediapipe --- main module
code: import mediapipe

solutions --- provides a suite of libraries for detecting hands, face, bodymarks for
better predection using ML & AI algorithms
code: mediapipe.solutions

hands --- submodule for hands in suite of libararies
code: mediapipe.solutions.hands

Hands() --- class to detect hands in the given input
code: mediapipe.solutions.hands.Hands(<parameters>)

HandLandmark[--point_name--] --- used to access each point on the hand, --
point_name-- for identification like THUMB_TIP, INDEX_FINGER_TIP etc..
code: mediapipe.solutions.hands.HandLandmark[--point_name--]

process(image) --- used to identigy the hand in the given image
code: mediapipe.solutions.hands.Hands().process(<image>)
```

```
drawing_utils --- function used to draw the points on the predected image
code: mediapipe.solutions.drawing_utils
'''
import cv2
import mediapipe as mp
import pyautogui
from math import dist
# getting the screen size using pyautogui module
def start():
   mp_hand_detector = mp.solutions.hands
   mp_draw = mp.solutions.drawing_utils
   screen_width, screen_height = pyautogui.size()
   input = cv2.VideoCapture(0)  # capturing the video using opencv module
   hand_detection = mp_hand_detector.Hands(max_num_hands=1,
min_detection_confidence=0.98)
   def position(point_name):
   # Get finger point number
      point_value = mp_hand_detector.HandLandmark[point_name].value
      # Get finger point position
      point_position_x = int((hand_landmarks.landmark[point_value].x *
frame_width)*screen_width/frame_width)#getting finger point x cordinates
      point_position_y = int((hand_landmarks.landmark[point_value].y *
frame_height)*screen_height/frame_height)#getting finger point y cordinates
      return ([point_position_x, point_position_y])
   while input.isOpened():
      _, image = input.read()  # reading input video from cv2 module
      frame_height, frame_width, _ = image.shape  # getting the dimensions of the
frame
      image = cv2.flip(image, 1)  # flipping the image
      image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
      results = hand_detection.process(image)  # making hand prediction of the image
      image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
      if results.multi_hand_landmarks:
```

hand_landmarks = results.multi_hand_landmarks[0] # *predicting landmarks of one hand only*

mp_draw.draw_landmarks(image, hand_landmarks, mp_hand_detector.HAND_CONNECTIONS) # *drawing the points of the hand on the image*

thumb_finger_tip = position("THUMB_TIP") # *getting the location of the thumb finger tip*

index_finger_tip = position("INDEX_FINGER_TIP") # *getting the location of the index finger tip*

index_finger_dip = position("INDEX_FINGER_DIP") # *getting the location of index finger dip*

middle_finger_tip = position("MIDDLE_FINGER_TIP") # *getting the location of middle finger tip*

middle_finger_dip = position("MIDDLE_FINGER_DIP") # *getting the location of middle finger dip*

ring_finger_dip = position("RING_FINGER_DIP") # *getting the location of ring finger dip*

pinky_finger_tip = position("PINKY_TIP") # *getting the location of pinky finger tip*

pinky_finger_dip = position("PINKY_DIP") # *getting the location of pinky finger dip*

thumb_index_dist = int(dist(index_finger_tip,thumb_finger_tip)) # *calculating the distance between the thumb and index finger tip*

#calculating distances for open palm

index_middle_dist = int(dist(middle_finger_dip, index_finger_tip)) # *calculating the distance between the middle finger dip and index finger tip*

middle_ring_dist = int(dist(ring_finger_dip, middle_finger_dip)) # *calculating the distance between the ring finger dip and middle finger dip*

ring_pinky_dist = int(dist(pinky_finger_tip, ring_finger_dip)) # *calculating the distance between the ring finger dip and pinky finger tip*

#calculating distances for closed palm

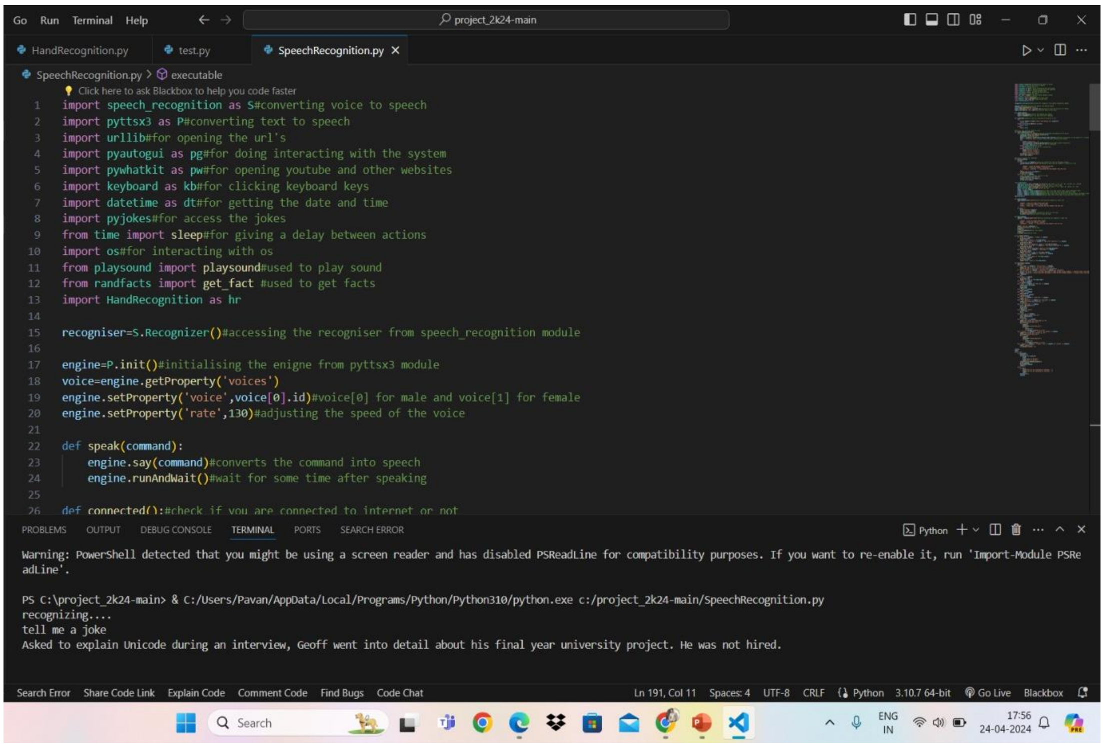
```
            index_tip_dip_dist = int(dist(index_finger_tip, index_finger_dip))  #
calculating the distance between the index finger dip and index finger tip
            middle_tip_dip_dist = int(dist(middle_finger_tip, middle_finger_dip))  #
calculating the distance between the middle finger tip and middle finger dip
            pinky_tip_dip_dist = int(dist(pinky_finger_tip, pinky_finger_dip))  #
calculating the distance between the pinky finger dip and pinky finger tip
            if thumb_index_dist >= 50 and thumb_index_dist <= 150 and
index_middle_dist >= 80:  # if the distance between the thumb finger and index finger
tips is less than 130 and grater than 50 decreasing the volume
                pyautogui.press('volumedown')
                print(thumb_index_dist)
            elif thumb_index_dist >= 151 and thumb_index_dist <= 300 and
index_middle_dist >= 81:  # if the distance between the thumb finger and index finger
tips is less than 250 and grater than 130 increasing the volume
                pyautogui.press('volumeup')
                print(thumb_index_dist)
            elif index_tip_dip_dist <= 50 and middle_tip_dip_dist <= 40 and
pinky_tip_dip_dist <= 40 and thumb_index_dist <= 100:
                print("closed palm")
                break
            elif index_middle_dist <= 80 and middle_ring_dist <= 90 and ring_pinky_dist
<= 80 and thumb_index_dist >= 230 and not (pinky_tip_dip_dist <= 20):
                print("open palm")
        cv2.imshow('Hand Detection', image)  # display the video on the screen
        if cv2.waitKey(5) & 0xFF == 27:  # close the output if we press Esc button
            break
    input.release()
    cv2.destroyAllWindows()
```

File - SpeechRecognition.py :

```
import speech_recognition as S#converting voice to speech

import pyttsx3 as P#converting text to speech

import urllib#for opening the url's

import pyautogui as pg#for doing interacting with the system

import pywhatkit as pw#for opening youtube and other websites

import keyboard as kb#for clicking keyboard keys

import datetime as dt#for getting the date and time

import pyjokes#for access the jokes

from time import sleep#for giving a delay between actions

import os#for interacting with os

from playsound import playsound#used to play sound

from randfacts import get_fact #used to get facts

import HandRecognition as hr

recogniser=S.Recognizer()#accessing the recogniser from speech_recognition module

engine=P.init()#initialising the enigne from pyttsx3 module

voice=engine.getProperty('voices')

engine.setProperty('voice',voice[0].id)#voice[0] for male and voice[1] for female

engine.setProperty('rate',130)#adjusting the speed of the voice

def speak(command):

    engine.say(command)#converts the command into speech

    engine.runAndWait()#wait for some time after speaking

def connected():#check if you are connected to internet or not

    try:
        urllib.request.urlopen('https://www.google.com',timeout=2)
        return True
    except urllib.error.URLError as error:
        return False
    except:
        return False
#function that converts speech to text
def voice_command_processor():
    with S.Microphone() as source:#accessing the primary microphone as the source
        recogniser.adjust_for_ambient_noise(source)
        playsound("sounds/open.wav")
        audio = recogniser.listen(source,phrase_time_limit=3)#listening continuously to
the speech for 4 seconds
        text = ''#creating empty string to store the text converted form speech
        try:
            print("recognizing....")
            text=recogniser.recognize_google(audio)#using google speech recognition
            text=text.lower()#converting the text to lowercase
        except S.UnknownValueError as e:
            print(e)#printing the error if occured any
        except S.RequestError as e:
            speak("service is down.")
            print("service is down")
            return("service is down")
        playsound("sounds/close.wav")

        return text.lower()

#function to search for something

def search(command):

    string=""

    if "search for" in command:#looking for "search for" key in the given
command

        string=command.split("search for")#exactly splitting the command at "search
for" key

        """

            command : could you please search for anything

            output : ['could you please','anything']

            string[1] : anything  --- accessing the 2nd element from the list

        """

        speak(f'Searching {string[1]}')

        pw.search(string[1])

    elif "search" in command:

        #same like above, instead "search" is used

        string=command.split("search")

        speak(f"Searching {string[1]}")

        pw.search(string[1])

class time_date():

    time=dt.datetime.now().strftime("%I %M %p")#to get time --- %I : hrs, %M :
minutes, %p : seconds

    time_24=dt.datetime.now().strftime("%H")#to get hours in 24h format
    date=dt.datetime.now().strftime("%d %B %Y")#to get date --- %d : Date, %B : Month, %Y : Year
    day=dt.datetime.now().strftime("%A")#to get day
    today = dt.date.today()
    y_date = today-dt.timedelta(days=1)#subtracting 1 day from current date to get yesterday's date
    t_date = today+dt.timedelta(days=1)#adding 1 day to current date to get tomorrow's date
    dy_date = today-dt.timedelta(days=2)#subtracting 2 days from current date to get day before yesterday's date
    dt_date = today+dt.timedelta(days=2)#adding 2 days from current date to get day after tomorrow's date
obj=time_date()
def connected():#check if you are connected to internet or not
    try:
        urllib.request.urlopen('https://www.google.com',timeout=2)
        return True
    except urllib.error.URLError as error:
        return False
    except:
        return False
#function that converts speech to text
def voice_command_processor():
    with S.Microphone() as source:#accessing the primary microphone as the source
        recogniser.adjust_for_ambient_noise(source)
        playsound("sounds/open.wav")
        audio = recogniser.listen(source,phrase_time_limit=3)#listening continuously to the speech for 4 seconds
        text = "#creating empty string to store the text converted form speech
        try:
            print("recognizing....")
            text=recogniser.recognize_google(audio)#using google speech recognition
            text=text.lower()#converting the text to lowercase
        except S.UnknownValueError as e:
            print(e)#printing the error if occured any
        except S.RequestError as e:
            speak("service is down.")
            print("service is down")
            return("service is down")
        playsound("sounds/close.wav")
        return text.lower()
#function to search for something
def play(command):
    song=command.split("play")#exactly splitting the command at "play" key
    """
        command : could you please play Alone song
        output : ['could you please','Alone song']
        song[1] : Alone song  --- accessing the 2nd element from the list
    """
    if song!="":
        print("playing "+song[1])
        speak("playing"+song[1])
        pw.playonyt(song[1])#open youtube and search for the song
        kb.press("space")#press enter to play the song
def connected():#check if you are connected to internet or not
    try:
        urllib.request.urlopen('https://www.google.com',timeout=2)
        return True
    except urllib.error.URLError as error:
        return False
    except:
        return False
#function that converts speech to text
def voice_command_processor():
    with S.Microphone() as source:#accessing the primary microphone as the source
        recogniser.adjust_for_ambient_noise(source)
        playsound("sounds/open.wav")
        audio = recogniser.listen(source,phrase_time_limit=3)#listening continuously to
the speech for 4 seconds
        text = ''#creating empty string to store the text converted form speech
        try:
            print("recognizing....")
            text=recogniser.recognize_google(audio)#using google speech recognition
            text=text.lower()#converting the text to lowercase
        except S.UnknownValueError as e:
            print(e)#printing the error if occured any
        except S.RequestError as e:
            speak("service is down.")
            print("service is down")
            return("service is down")
        playsound("sounds/close.wav")
        return text.lower()
#function to search for something
def open(command):
    command = command.split("open")#exactly splitting the command at "open" key
    """
        command : could you please open chrome
        output : ['could you please','chrome']
        command[1] : chrome  --- accessing the 2nd element from the list
    """
    speak(f'opening {command[1]}')
    pg.press("win")#press the win key
    sleep(0.5)
    kb.write(command[1])#enter the command
    sleep(0.5)
    kb.press("enter")#click enter
def y_t_date(command):
    if "today's" in command or "today" in command:
        print(obj.date)
        speak("today's date is "+str(obj.date))
    elif "day after tomorrow" in command or "day after tomorrow's" in command:
        print(obj.dt_date)
        speak("day after tomoerrow's date is "+str(obj.dt_date))
    elif "day before yesterday" in command or "day before yesterday's" in command:
        print(obj.dy_date)
        speak("day before yesterday's date is "+str(obj.dy_date))
    elif "yesterday's" in command or "yesterday" in command:
        print(obj.y_date)
        speak("yesterday's date is "+str(obj.y_date))
    elif "tomorrow's" in command or "tomorrow" in command:
        print(obj.t_date)
        speak("tomorrow's date is "+str(obj.t_date))
    elif "date" in command:
        print(obj.date)
        speak("today's date is "+str(obj.date))
def executable(command):
    num=[2,1]
    if "your name" in command or "who are you" in command:
        speak("My name is Elite, i'm your voice assistant")
        print("My name is The Elite, i'm your voice assistent")
    if "tell me about" in command or "describe" in command and "yourself":
        print("I am Elite, a voice assistant developed in 2024 by BITS Vizag students. I execute tasks like opening apps, searching, and playing music.")
        speak("I am Elite, a voice assistant developed in 2024 by BITS Vyzaag students. I execute tasks like opening apps, searching, and playing music.")
    elif command=="":
        pass
    elif "time" in command:
        print(obj.time)
        speak("the time is "+str(obj.time))
    elif "date" in command:
        y_t_date(command)
    elif "joke" in command and "tell me" in command:
        joke=pyjokes.get_joke()
        print(joke)
        speak(joke)
    elif "open" in command:
        open(command)
    elif "play" in command:
        play(command)
    elif "search" in command:
        search(command)
    elif "bye" in command or "good bye" in command:
        speak("Bye")
    elif "hand gestures" in command and "activate" in command:
```

```
    speak("Activating Hand Gestures...")
    hr.start()
elif "say something" in command:
    fact = get_fact(False)
    print(fact)
    speak(fact)
elif command == "exit":
    print("Okay")
    speak("okay")
    exit()
elif "what is" in command:
    text=command.split("what is")
    op=pw.info(text[1])
    print(op)
elif "shut down" in command:
    speak("The system will Shut down in, 3")
    playsound("sounds/beep.wav")
    for i in num:
        speak(i)
        playsound("sounds/beep.wav")
        if i==1:
            print(True)
            os.system("shutdown /s /t 5")
elif "restart" in command or "restart the system" in command:
        speak("The system will restart in, 3")
        playsound("sounds/beep.wav")
        for i in num:
            speak(i)
            playsound("sounds/beep.wav")
            if i==1:
                print(True)
                os.system("shutdown /r /t 5")
    elif "logout" in command or ("logout from" in command and "system" in command):
        speak("Logging out")
        os.system("shutdown -l")
count=1
num=0
while True:
    if connected():
        boolean=True
        if boolean and count==0:
            num=0
            print("Back to Online")
            speak("Back to online")
        command=voice_command_processor()
        print(command)
        executable(command)
    sleep(1)
else:
    if num<2:
        print("You are not connected to internet...")
        speak("You are not connected to internet...")
        num+=1
    boolean=False
    count=0
```

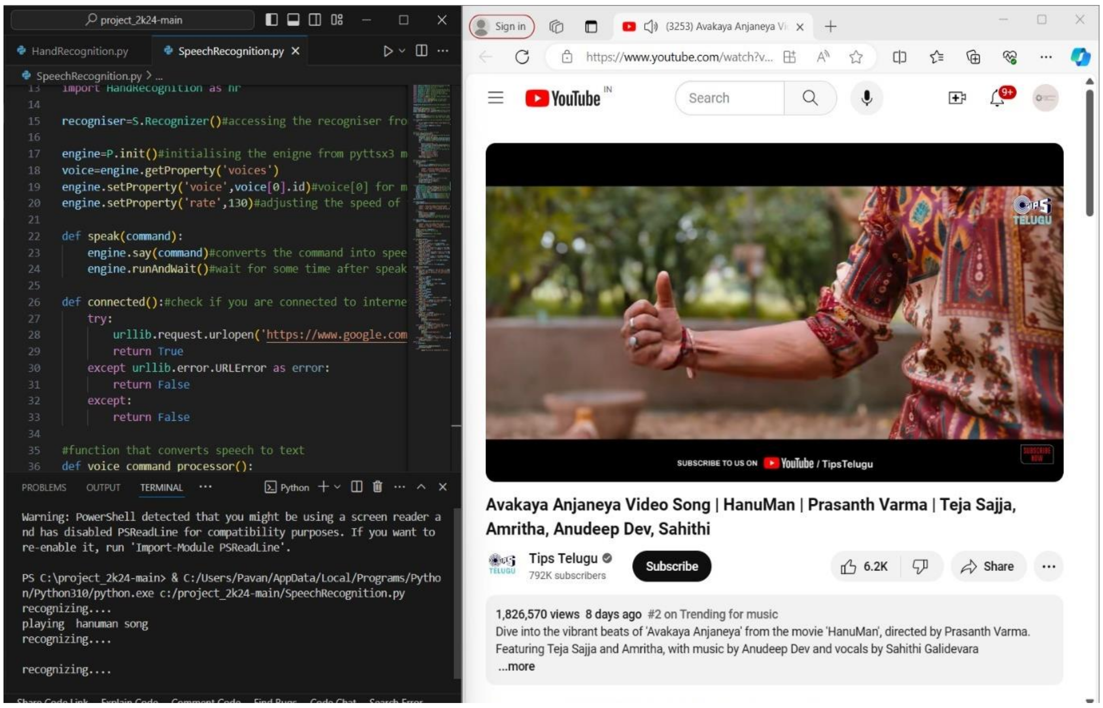
5.3. Results -



```
import speech_recognition as S#converting voice to speech
import pyttsx3 as P#converting text to speech
import urllib#for opening the url's
import pyautogui as pg#for doing interacting with the system
import pywhatkit as pw#for opening youtube and other websites
import keyboard as kb#for clicking keyboard keys
import datetime as dt#for getting the date and time
import pyjokes#for access the jokes
from time import sleep#for giving a delay between actions
import os#for interacting with os
from playsound import playsound#used to play sound
from randfacts import get_fact #used to get facts
import HandRecognition as hr

recogniser=S.Recognizer()#accessing the recogniser from speech_recognition module

engine=P.init()#initialising the enigne from pyttsx3 module
voice=engine.getProperty('voices')
engine.setProperty('voice',voice[0].id)#voice[0] for male and voice[1] for female
engine.setProperty('rate',130)#adjusting the speed of the voice

def speak(command):
    engine.say(command)#converts the command into speech
    engine.runAndWait()#wait for some time after speaking

def connected():#check if you are connected to internet or not
```

```
Warning: PowerShell detected that you might be using a screen reader and has disabled PSReadLine for compatibility purposes. If you want to re-enable it, run 'Import-Module PSRe
adLine'.

PS C:\project_2k24-main> & C:/Users/Pavan/AppData/Local/Programs/Python/Python310/python.exe c:/project_2k24-main/SpeechRecognition.py
recognizing....
tell me a joke
Asked to explain Unicode during an interview, Geoff went into detail about his final year university project. He was not hired.
```

Fig 5.3.1: Ask the module to tell me a joke.



```
import HandRecognition as hr

recogniser=S.Recognizer()#accessing the recogniser fro

engine=P.init()#initialising the enigne from pyttsx3 m
voice=engine.getProperty('voices')
engine.setProperty('voice',voice[0].id)#voice[0] for m
engine.setProperty('rate',130)#adjusting the speed of

def speak(command):
    engine.say(command)#converts the command into spee
    engine.runAndWait()#wait for some time after speak

def connected():#check if you are connected to interne
    try:
        urllib.request.urlopen('https://www.google.com
        return True
    except urllib.error.URLError as error:
        return False
    except:
        return False

#function that converts speech to text
def voice_command_processor():
```

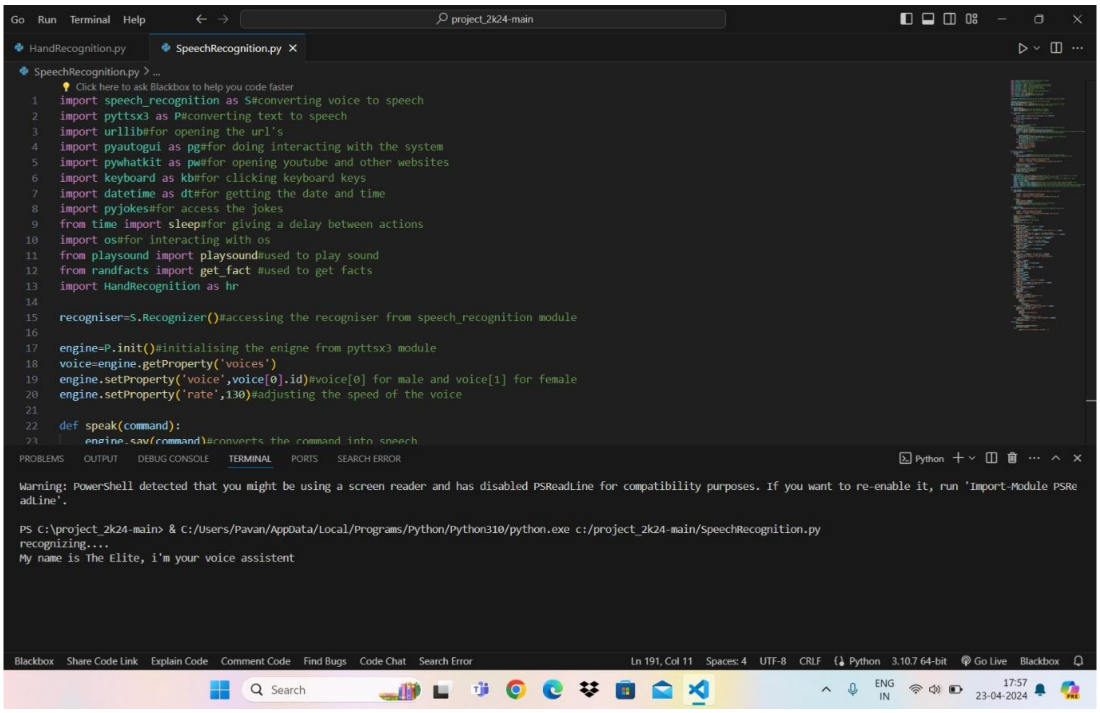
```
Warning: PowerShell detected that you might be using a screen reader a
nd has disabled PSReadLine for compatibility purposes. If you want to
re-enable it, run 'Import-Module PSReadLine'.

PS C:\project_2k24-main> & C:/Users/Pavan/AppData/Local/Programs/Pytho
n/Python310/python.exe c:/project_2k24-main/SpeechRecognition.py
recognizing....
playing  hanuman song
recognizing....

recognizing....
```

Avakaya Anjaneya Video Song | HanuMan | Prasanth Varma | Teja Sajja, Amritha, Anudeep Dev, Sahithi

Tips Telugu
792K subscribers

1,826,570 views 8 days ago #2 on Trending for music
Dive into the vibrant beats of 'Avakaya Anjaneya' from the movie 'HanuMan', directed by Prasanth Varma. Featuring Teja Sajja and Amritha, with music by Anudeep Dev and vocals by Sahithi Galidevara
...more

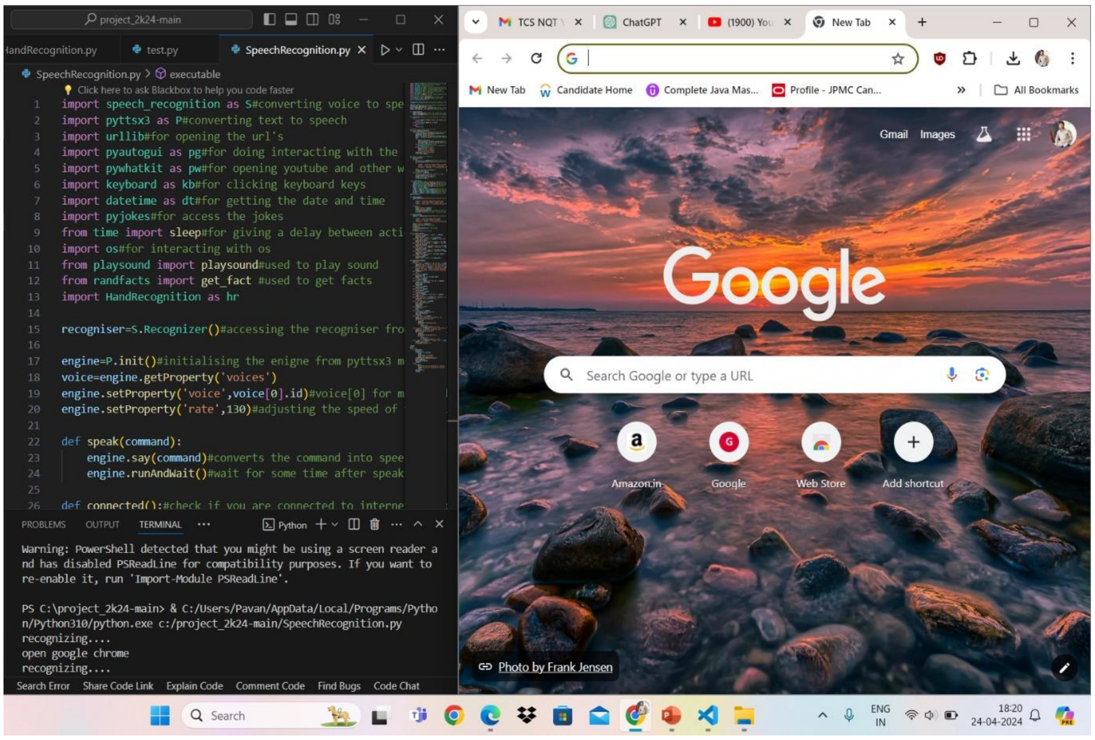
Fig 5.3.2: Playing a video in YouTube.



```python
import speech_recognition as S#converting voice to speech
import pyttsx3 as P#converting text to speech
import urllib#for opening the url's
import pyautogui as pg#for doing interacting with the system
import pywhatkit as pw#for opening youtube and other websites
import keyboard as kb#for clicking keyboard keys
import datetime as dt#for getting the date and time
import pyjokes#for access the jokes
from time import sleep#for giving a delay between actions
import os#for interacting with os
from playsound import playsound#used to play sound
from randfacts import get_fact #used to get facts
import HandRecognition as hr

recogniser=S.Recognizer()#accessing the recogniser from speech_recognition module

engine=P.init()#initialising the enigne from pyttsx3 module
voice=engine.getProperty('voices')
engine.setProperty('voice',voice[0].id)#voice[0] for male and voice[1] for female
engine.setProperty('rate',130)#adjusting the speed of the voice

def speak(command):
    engine.say(command)#converts the command into speech
```

```
Warning: PowerShell detected that you might be using a screen reader and has disabled PSReadLine for compatibility purposes. If you want to re-enable it, run 'Import-Module PSRe
adLine'.

PS C:\project_2k24-main> & C:/Users/Pavan/AppData/Local/Programs/Python/Python310/python.exe c:/project_2k24-main/SpeechRecognition.py
recognizing....
My name is The Elite, i'm your voice assistent
```

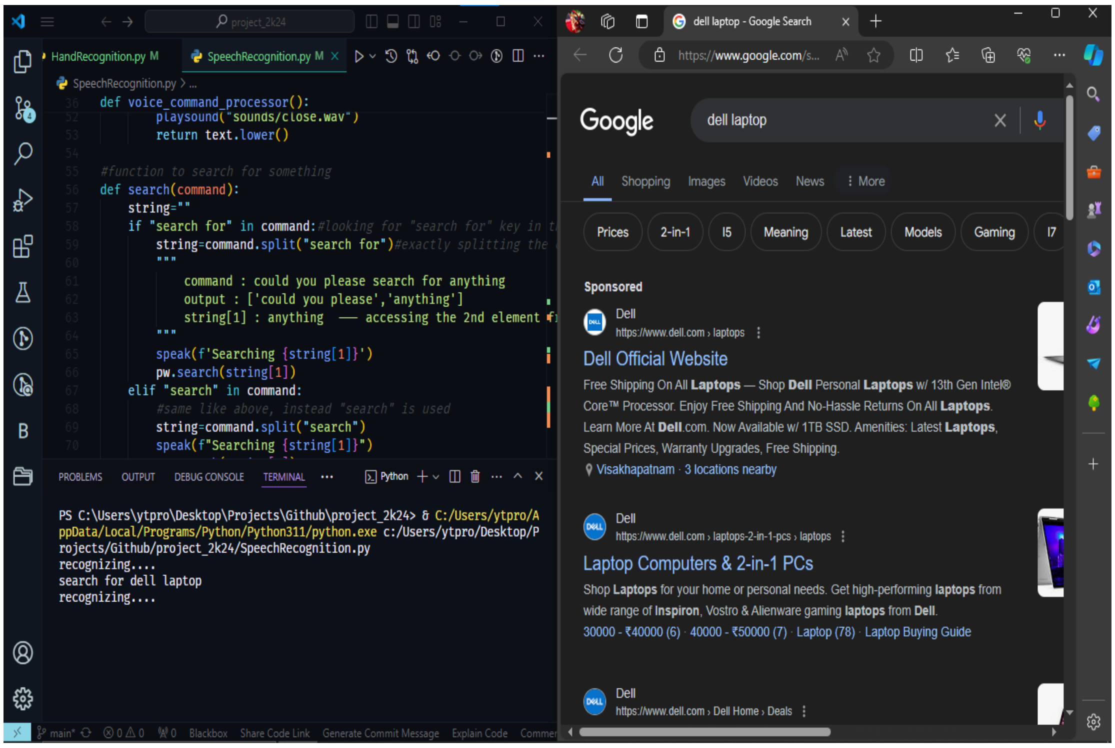
Fig 5.3.3: Ask the module to describe herself.



```python
import speech_recognition as S#converting voice to spe
import pyttsx3 as P#converting text to speech
import urllib#for opening the url's
import pyautogui as pg#for doing interacting with the
import pywhatkit as pw#for opening youtube and other w
import keyboard as kb#for clicking keyboard keys
import datetime as dt#for getting the date and time
import pyjokes#for access the jokes
from time import sleep#for giving a delay between acti
import os#for interacting with os
from playsound import playsound#used to play sound
from randfacts import get_fact #used to get facts
import HandRecognition as hr

recogniser=S.Recognizer()#accessing the recogniser fro

engine=P.init()#initialising the enigne from pyttsx3 m
voice=engine.getProperty('voices')
engine.setProperty('voice',voice[0].id)#voice[0] for m
engine.setProperty('rate',130)#adjusting the speed of

def speak(command):
    engine.say(command)#converts the command into spee
    engine.runAndWait()#wait for some time after speak

def connected():#check if you are connected to interne
```

```
Warning: PowerShell detected that you might be using a screen reader a
nd has disabled PSReadLine for compatibility purposes. If you want to
re-enable it, run 'Import-Module PSReadLine'.

PS C:\project_2k24-main> & C:/Users/Pavan/AppData/Local/Programs/Pytho
n/Python310/python.exe c:/project_2k24-main/SpeechRecognition.py
recognizing....
open google chrome
recognizing....
```

Fig 5.3.4: Opening external Applications with voice.



```python
def voice_command_processor():
        playsound("sounds/close.wav")
        return text.lower()

#function to search for something
def search(command):
    string=""
    if "search for" in command:#looking for "search for" key in t
        string=command.split("search for")#exactly splitting the
        """
            command : could you please search for anything
            output : ['could you please','anything']
            string[1] : anything  —— accessing the 2nd element fi
        """
        speak(f'Searching {string[1]}')
        pw.search(string[1])
    elif "search" in command:
        #same like above, instead "search" is used
        string=command.split("search")
        speak(f"Searching {string[1]}")
```

```
PS C:\Users\ytpro\Desktop\Projects\Github\project_2k24> & C:/Users/ytpro/A
ppData/Local/Programs/Python/Python311/python.exe c:/Users/ytpro/Desktop/P
rojects/Github/project_2k24/SpeechRecognition.py
recognizing....
search for dell laptop
recognizing....
```

dell laptop

Sponsored

Dell
https://www.dell.com › laptops

Dell Official Website

Free Shipping On All **Laptops** — Shop **Dell** Personal **Laptops** w/ 13th Gen Intel® Core™ Processor. Enjoy Free Shipping And No-Hassle Returns On All **Laptops**. Learn More At **Dell**.com. Now Available w/ 1TB SSD. Amenities: Latest **Laptops**, Special Prices, Warranty Upgrades, Free Shipping.

Visakhapatnam · 3 locations nearby

Dell
https://www.dell.com › laptops-2-in-1-pcs › laptops

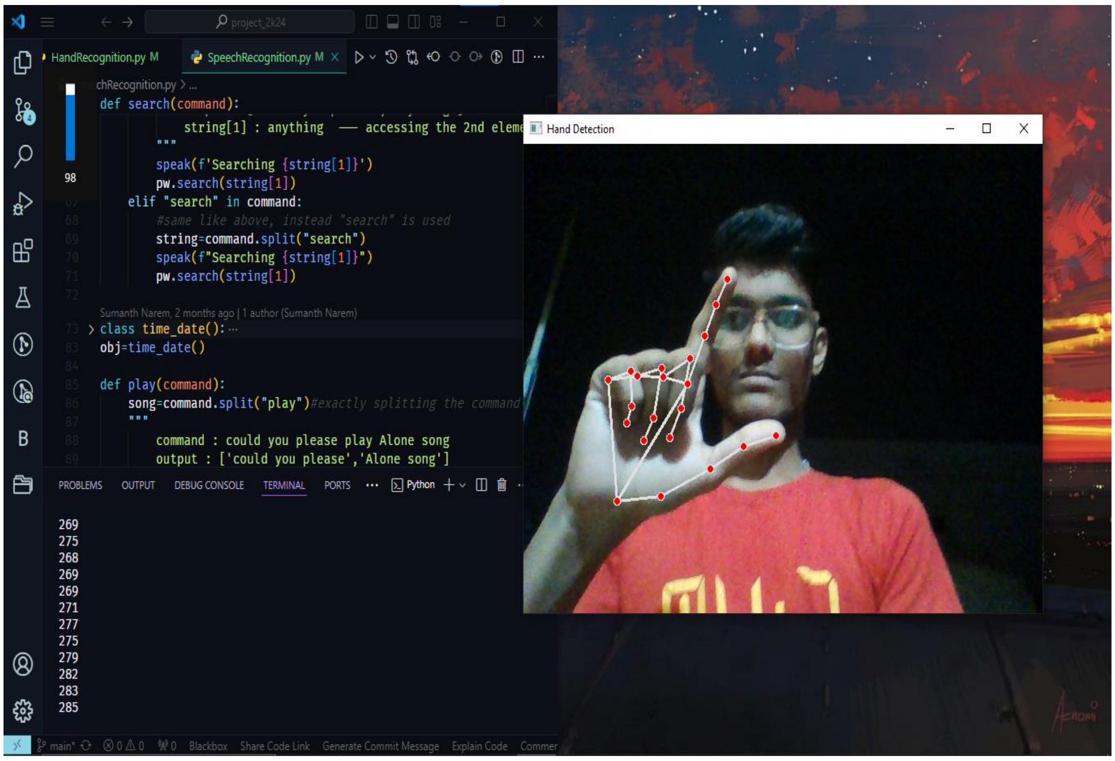
Laptop Computers & 2-in-1 PCs

Shop **Laptops** for your home or personal needs. Get high-performing **laptops** from wide range of **Inspiron**, Vostro & Alienware gaming **laptops** from **Dell**.

30000 - ₹40000 (6) · 40000 - ₹50000 (7) · Laptop (78) · Laptop Buying Guide

Dell
https://www.dell.com › Dell Home › Deals

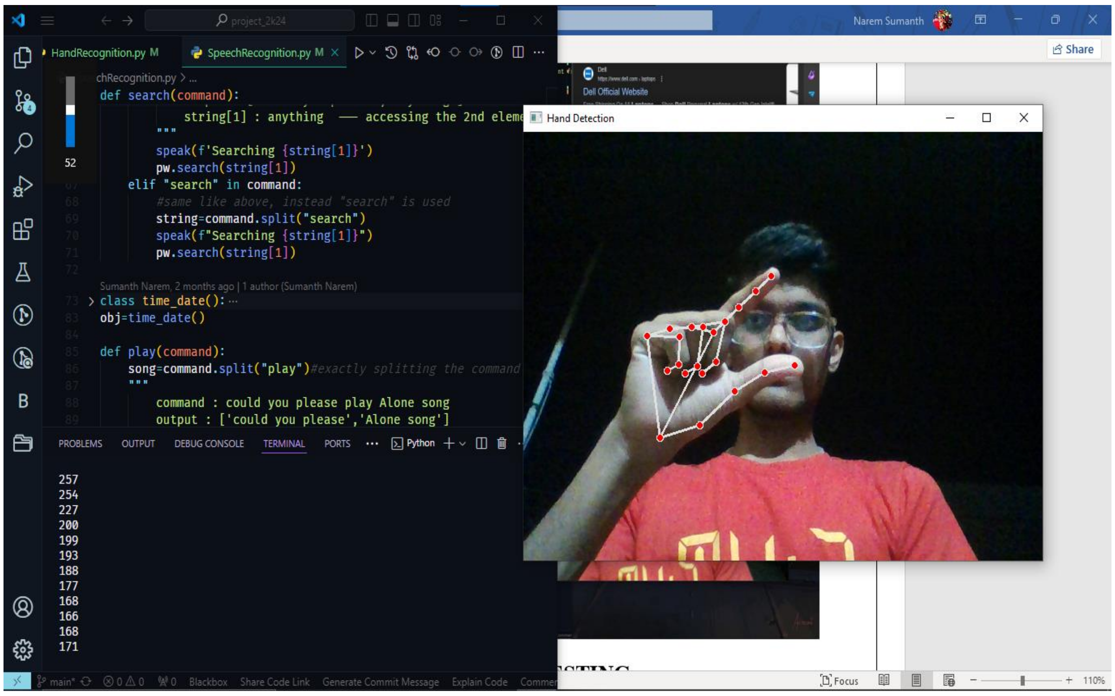
Fig 5.3.5: Searching specific topic.



```python
def search(command):
            string[1] : anything  —— accessing the 2nd eleme
        """
        speak(f'Searching {string[1]}')
        pw.search(string[1])
    elif "search" in command:
        #same like above, instead "search" is used
        string=command.split("search")
        speak(f"Searching {string[1]}")
        pw.search(string[1])

Sumanth Narem, 2 months ago | 1 author (Sumanth Narem)
class time_date():...
obj=time_date()

def play(command):
    song=command.split("play")#exactly splitting the command
    """
        command : could you please play Alone song
        output : ['could you please','Alone song']
```

```
269
275
268
269
269
271
277
275
279
282
283
285
```

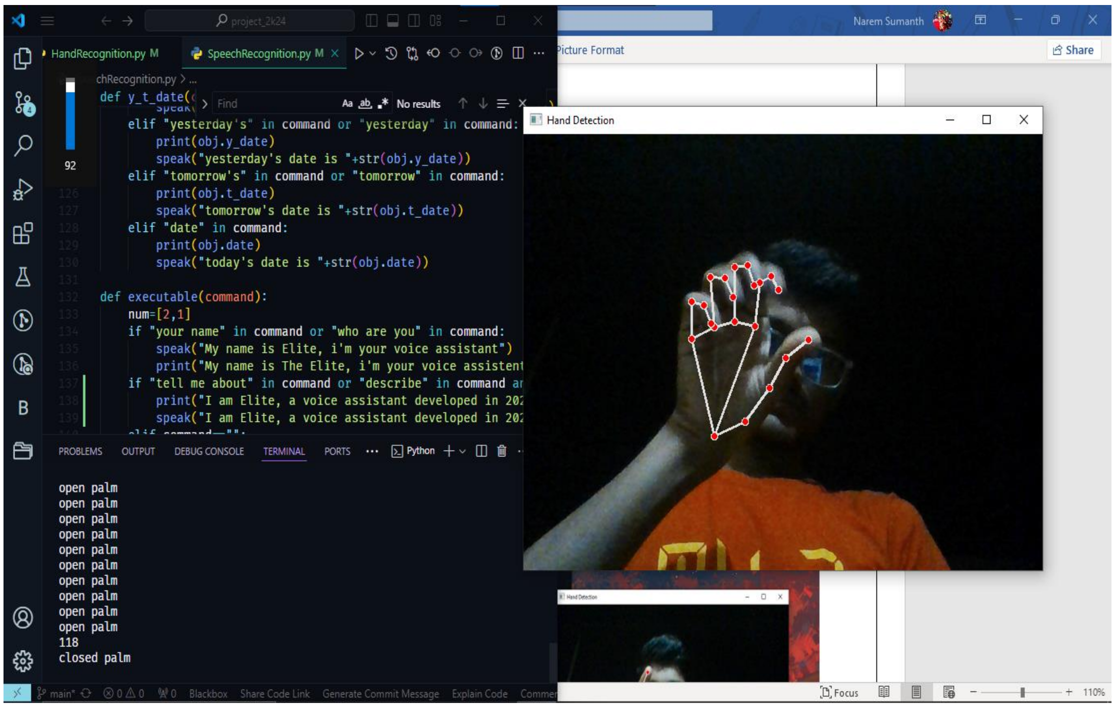
Fig 5.3.6: Volume control using Hand Gestures(Increasing Volume).



```python
def search(command):
            string[1] : anything  —— accessing the 2nd eleme
        """
        speak(f'Searching {string[1]}')
        pw.search(string[1])
    elif "search" in command:
        #same like above, instead "search" is used
        string=command.split("search")
        speak(f"Searching {string[1]}")
        pw.search(string[1])

class time_date(): ...
obj=time_date()

def play(command):
    song=command.split("play")#exactly splitting the command
    """
        command : could you please play Alone song
        output : ['could you please','Alone song']
```

```
257
254
227
200
199
193
188
177
168
166
168
171
```

Fig 5.3.7: *Volume control using Hand Gestures(Decreasing Volume).*



```python
def y_t_date(
    elif "yesterday's" in command or "yesterday" in command:
        print(obj.y_date)
        speak("yesterday's date is "+str(obj.y_date))
    elif "tomorrow's" in command or "tomorrow" in command:
        print(obj.t_date)
        speak("tomorrow's date is "+str(obj.t_date))
    elif "date" in command:
        print(obj.date)
        speak("today's date is "+str(obj.date))

def executable(command):
    num=[2,1]
    if "your name" in command or "who are you" in command:
        speak("My name is Elite, i'm your voice assistant")
        print("My name is The Elite, i'm your voice assistent
    if "tell me about" in command or "describe" in command ar
        print("I am Elite, a voice assistant developed in 202
        speak("I am Elite, a voice assistant developed in 202
```

```
open palm
open palm
open palm
open palm
open palm
open palm
open palm
open palm
open palm
open palm
118
closed palm
```

Fig 5.3.8: Deactivating Hand Gestures when it detect closed palm.

6. TESTING

6.1 Introduction to testing

Testing is the major quantity control measure employed during software development. Its basic function is to detect errors in the software. During requirement analysis and design, the output is a document that is usually textual and non-executable. After the coding phase, computer programs are available that can be executed for testing purposes. This implies that testing not only has to uncover errors introduced during coding, but also uncovers requirement, design or coding errors in the programs. Consequently, different levels of testing are employed.

These different levels of testing attempt to detect different types of faults.

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

Unit Testing: The first level of testing is "unit testing". In these different modules are tested against the specifications produced during design for the modules. Unit testing is essentially for verification of the code produced during the coding phase, and hence the goal is to test the internal logic of the modules.

Integration Testing: In this, many tested modules are combined into sub-systems, which are then tested. The goal is to see if the modules can be integrated properly, the emphasis being on testing interfaces between modules. This testing activity can be considered as testing the design, and hence the emphasis on testing module. interactions.

System Testing: Here the entire software system is tested. The reference document for this process is the requirements document, and the goal is to see if the software meets it requirements.

Acceptance Testing: Acceptance testing is sometimes performed with realistic data of the client to demonstrate that the software is working satisfactorily. Testing here focuses on the external behaviour of the system, the internal logic of the system is not emphasized.

1.Types of testing

To accomplish task software is done under two categories of test case design techniques Functional/ Black box Testing Structural/ White box Testing.

Functional testing: In functional testing the structure of the program is not considered. Test cases are decided solely on the basis of the requirements or specifications of the program or module and the internals of the module or the program are not considered for selection of test cases.

Structural testing: Structural testing is concerned with testing the implementation of the program. The intention of the structural testing is not to execute all the different input and output conditions, but to exercise the different programming and data structures used in the program. This testing is otherwise called as the "white box testing".

2.Test case specification

The test plan focuses on how the testing for the project will proceed, which units will be tested, and what approaches are to be used during the various stages of testing. Test case specifications have to be done separately for each unit. Based on the approach specified in the test plan, first the features to be tested for this unit must be determined. The overall approach stated in the plan is refined into specific test techniques that should be followed and into the criteria to be used for evaluation. Test case specification gives, for each unit to be tested, all test cases, inputs to be used in the test cases, conditions being tested by the test case, and outputs expected for those test cases. Test case specification is a major activity in the testing process.

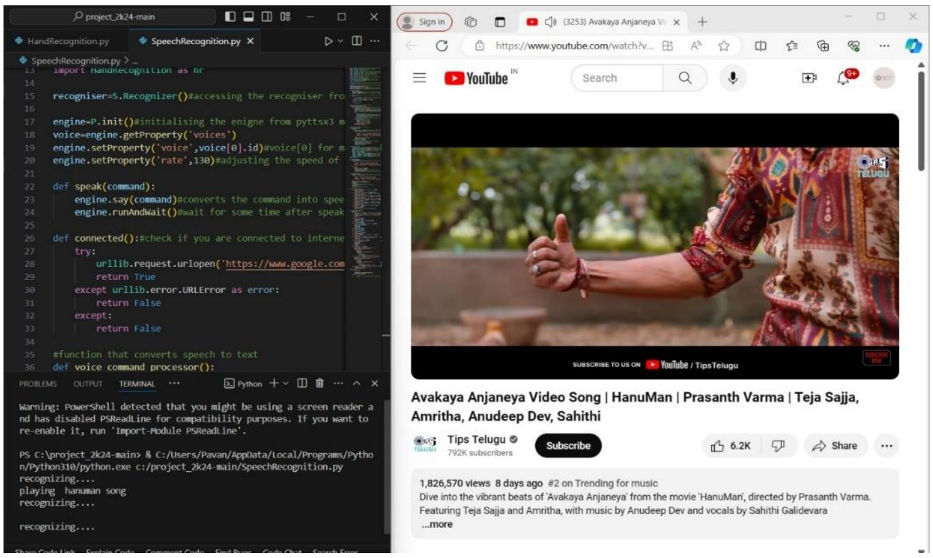
3.Testing Methodologies

- **Black box Testing:** It is the testing process in which tester can perform testing on an application without having any internal structural knowledge of application. Usually Test Engineers are involved in the black box testing. Black-Box Testing alludes to tests that are conducted at the software interface.

- **White box Testing:** It is the testing process in which tester can perform testing on an application with having internal structural knowledge. Usually the Developers are involved in white box testing.

6.2 Sample Test Cases

1. **Playing Songs:**
   Developing a system to play songs based on user-requested song names using speech recognition and integration with music libraries.



```
import HandRecognition as hr

recogniser=S.Recognizer()#accessing the recogniser fro

engine=P.init()#initialising the enigne from pyttsx3 m
voice=engine.getProperty('voices')
engine.setProperty('voice',voice[0].id)#voice[0] for m
engine.setProperty('rate',130)#adjusting the speed of

def speak(command):
    engine.say(command)#converts the command into spee
    engine.runAndWait()#wait for some time after speak

def connected():#check if you are connected to interne
    try:
        urllib.request.urlopen('https://www.google.com
        return True
    except urllib.error.URLError as error:
        return False
    except:
        return False

#function that converts speech to text
def voice command processor():
```

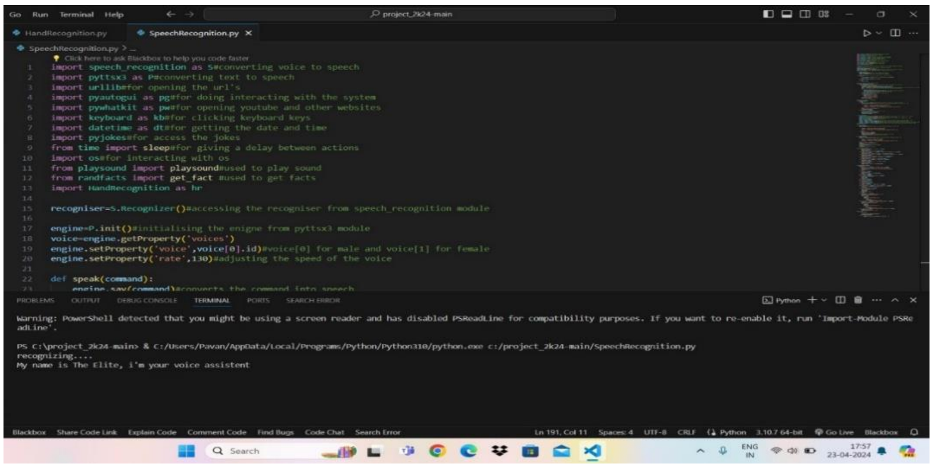
```
Warning: PowerShell detected that you might be using a screen reader a
nd has disabled PSReadLine for compatibility purposes. If you want to
re-enable it, run 'Import-Module PSReadLine'.

PS C:\project_2k24-main> & C:/Users/Pavan/AppData/Local/Programs/Pytho
n/Python310/python.exe c:/project_2k24-main/SpeechRecognition.py
recognizing....
playing  hanuman song
recognizing....

recognizing....
```

Avakaya Anjaneya Video Song | HanuMan | Prasanth Varma | Teja Sajja, Amritha, Anudeep Dev, Sahithi

1,826,570 views 8 days ago #2 on Trending for music
Dive into the vibrant beats of 'Avakaya Anjaneya' from the movie 'HanuMan', directed by Prasanth Varma. Featuring Teja Sajja and Amritha, with music by Anudeep Dev and vocals by Sahithi Galidevara
...more

Fig 6.2.1: Playing a video in YouTube.

2. **Describe about the System:**
   An innovative model integrating speech and gesture recognition, such that it can able to describe about herself.



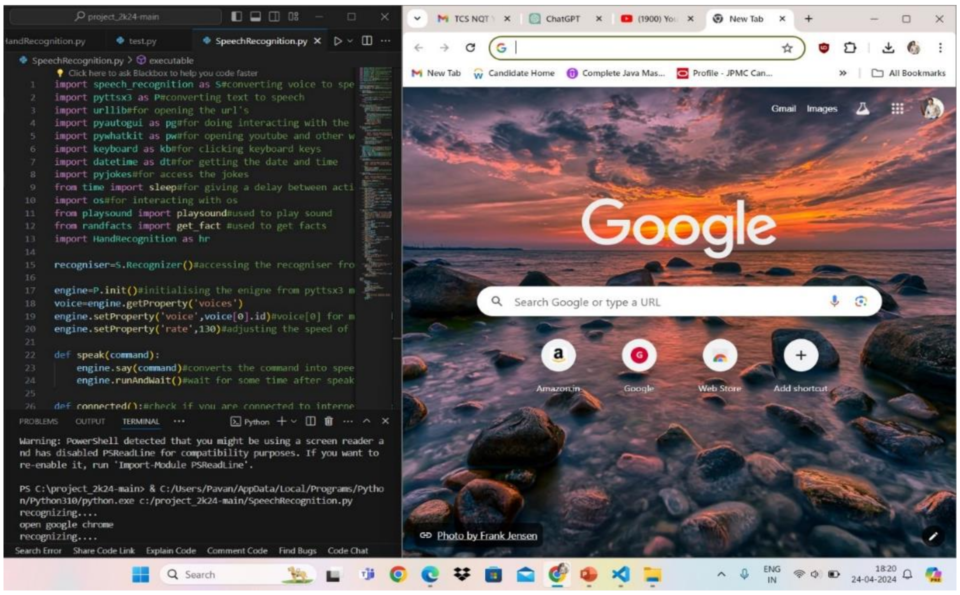
```
import speech_recognition as S#converting voice to speech
import pyttsx3 as P#converting text to speech
import urllib#for opening the url's
import pyautogui as pg#for doing interacting with the system
import pywhatkit as pw#for opening youtube and other websites
import keyboard as kb#for clicking keyboard keys
import datetime as dt#for getting the date and time
import pyjokes#for access the jokes
from time import sleep#for giving a delay between actions
import os#for interacting with os
from playsound import playsound#used to play sound
from randfacts import get_fact #used to get facts
import HandRecognition as hr

recogniser=S.Recognizer()#accessing the recogniser from speech_recognition module

engine=P.init()#initialising the enigne from pyttsx3 module
voice=engine.getProperty('voices')
engine.setProperty('voice',voice[0].id)#voice[0] for male and voice[1] for female
engine.setProperty('rate',130)#adjusting the speed of the voice

def speak(command):
    engine.say(command)#converts the command into speech
```

```
Warning: PowerShell detected that you might be using a screen reader and has disabled PSReadLine for compatibility purposes. If you want to re-enable it, run 'Import-Module PSRe
adLine'.

PS C:\project_2k24-main> & C:/Users/Pavan/AppData/Local/Programs/Python/Python310/python.exe c:/project_2k24-main/SpeechRecognition.py
recognizing....
My name is The Elite, i'm your voice assistent
```

Fig 6.2.2: Ask the module to describe herself.

3. Opening External Applications:

The project aims to seamlessly integrate speech and gesture recognition, allowing users to execute commands and open desired applications effortlessly.



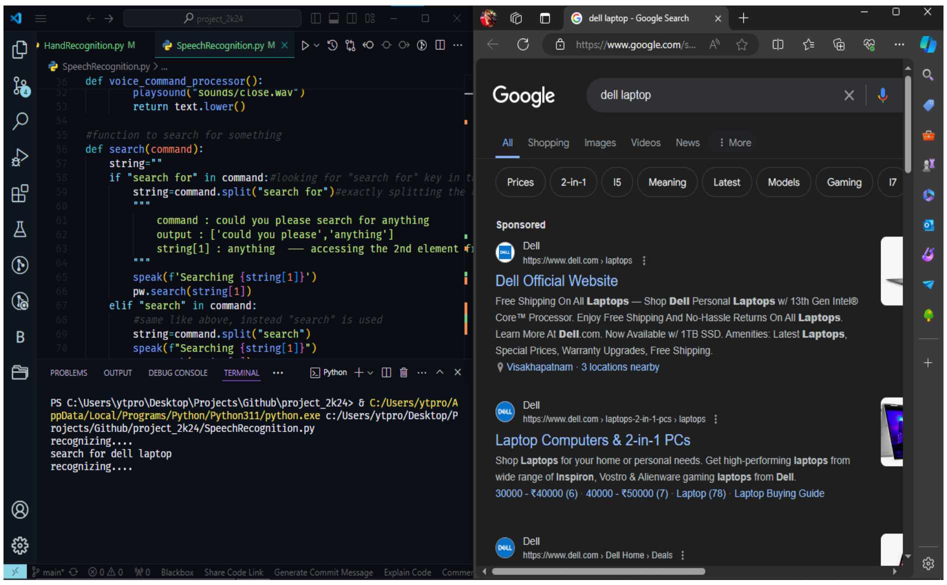
```
import speech_recognition as S#converting voice to spe
import pyttsx3 as P#converting text to speech
import urllib#for opening the url's
import pyautogui as pg#for doing interacting with the
import pywhatkit as pw#for opening youtube and other w
import keyboard as kb#for clicking keyboard keys
import datetime as dt#for getting the date and time
import pyjokes#for access the jokes
from time import sleep#for giving a delay between acti
import os#for interacting with os
from playsound import playsound#used to play sound
from randfacts import get_fact #used to get facts
import HandRecognition as hr

recogniser=S.Recognizer()#accessing the recogniser fro

engine=P.init()#initialising the enigne from pyttsx3 m
voice=engine.getProperty('voices')
engine.setProperty('voice',voice[0].id)#voice[0] for m
engine.setProperty('rate',130)#adjusting the speed of

def speak(command):
    engine.say(command)#converts the command into spee
    engine.runAndWait()#wait for some time after speak

def connected():#check if you are connected to interne
```

```
Warning: PowerShell detected that you might be using a screen reader a
nd has disabled PSReadLine for compatibility purposes. If you want to
re-enable it, run 'Import-Module PSReadLine'.

PS C:\project_2k24-main> & C:/Users/Pavan/AppData/Local/Programs/Pytho
n/Python310/python.exe c:/project_2k24-main/SpeechRecognition.py
recognizing....
open google chrome
recognizing....
```

Fig 6.2.3: Opening external Applications with voice.

4. Search for something:

In addition, the system can search for content in the web browser, further enhancing its functionality and usability for diverse tasks.



```
def voice_command_processor():
            playsound("sounds/close.wav")
            return text.lower()

#function to search for something
def search(command):
    string=""
    if "search for" in command:#looking for "search for" key in t
        string=command.split("search for")#exactly splitting the
        """
            command : could you please search for anything
            output : ['could you please','anything']
            string[1] : anything  —— accessing the 2nd element f
        """
        speak(f'Searching {string[1]}')
        pw.search(string[1])
    elif "search" in command:
        #same like above, instead "search" is used
        string=command.split("search")
        speak(f"Searching {string[1]}")
```

```
PS C:\Users\ytpro\Desktop\Projects\Github\project_2k24> & C:/Users/ytpro/A
ppData/Local/Programs/Python/Python311/python.exe c:/Users/ytpro/Desktop/P
rojects/Github/project_2k24/SpeechRecognition.py
recognizing....
search for dell laptop
recognizing....
```

Sponsored
Dell
https://www.dell.com › laptops
Dell Official Website
Free Shipping On All Laptops — Shop Dell Personal Laptops w/ 13th Gen Intel® Core™ Processor. Enjoy Free Shipping And No-Hassle Returns On All Laptops. Learn More At Dell.com. Now Available w/ 1TB SSD. Amenities: Latest Laptops, Special Prices, Warranty Upgrades, Free Shipping.
Visakhapatnam · 3 locations nearby

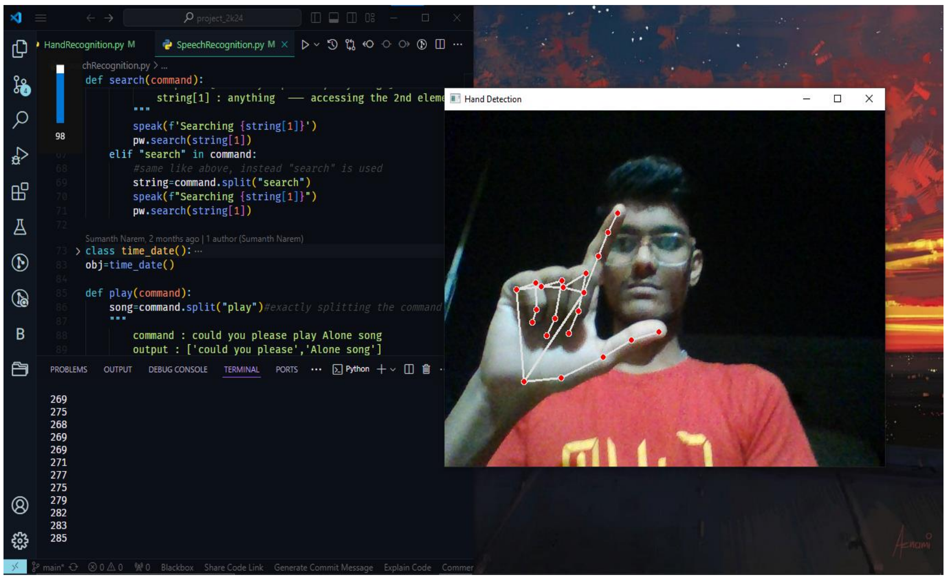
Dell
https://www.dell.com › laptops-2-in-1-pcs › laptops
Laptop Computers & 2-in-1 PCs
Shop Laptops for your home or personal needs. Get high-performing laptops from wide range of Inspiron, Vostro & Alienware gaming laptops from Dell.
30000 - ₹40000 (6) · 40000 - ₹50000 (7) · Laptop (78) · Laptop Buying Guide

Dell
https://www.dell.com › Dell Home › Deals

Fig 6.2.4: Searching specific topic.

5. Volume Control(Volume Up):

After activating hand gestures, users can control volume using thumb and index finger gestures, enhancing interaction versatility and user experience.



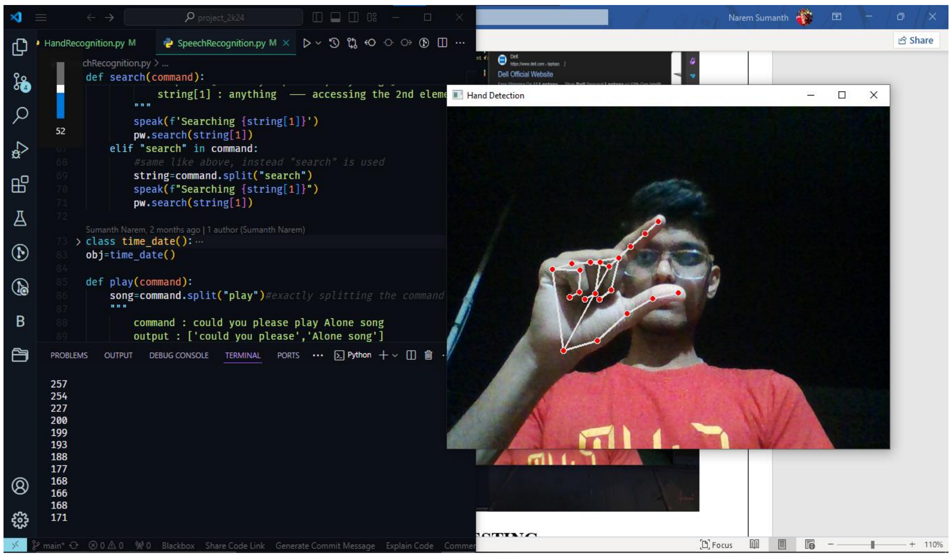
```
def search(command):
            string[1] : anything  —— accessing the 2nd eleme
        """
        speak(f'Searching {string[1]}')
        pw.search(string[1])
    elif "search" in command:
        #same like above, instead "search" is used
        string=command.split("search")
        speak(f"Searching {string[1]}")
        pw.search(string[1])

Sumanth Narem, 2 months ago | 1 author (Sumanth Narem)
class time_date():...
obj=time_date()

def play(command):
    song=command.split("play")#exactly splitting the command
    """
        command : could you please play Alone song
        output : ['could you please','Alone song']
```

```
269
275
268
269
269
271
277
275
279
282
283
285
```

Fig 6.2.5: Volume control using Hand Gestures(Increasing Volume).

6. Volume Control(Volume Down):

After activating hand gestures, users can control volume using thumb and index finger gestures, enhancing interaction versatility and user experience.



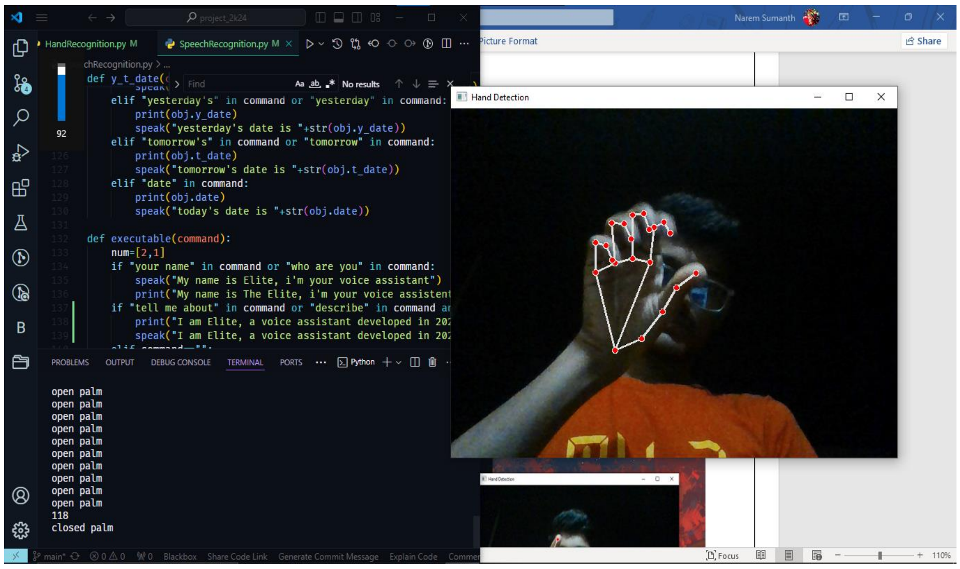
```
def search(command):
            string[1] : anything  —— accessing the 2nd eleme
        """
        speak(f'Searching {string[1]}')
        pw.search(string[1])
    elif "search" in command:
        #same like above, instead "search" is used
        string=command.split("search")
        speak(f"Searching {string[1]}")
        pw.search(string[1])

Sumanth Narem, 2 months ago | 1 author (Sumanth Narem)
class time_date():...
obj=time_date()

def play(command):
    song=command.split("play")#exactly splitting the command
    """
        command : could you please play Alone song
        output : ['could you please','Alone song']
```

```
257
254
227
200
199
193
188
177
168
166
168
171
```

Fig 6.2.6: Volume control using Hand Gestures(Decreasing Volume).

7. Deactivating Hand Gestures:

Upon detecting a closed palm, the system deactivates hand gestures, ensuring seamless transition to voice-only interaction mode for users.



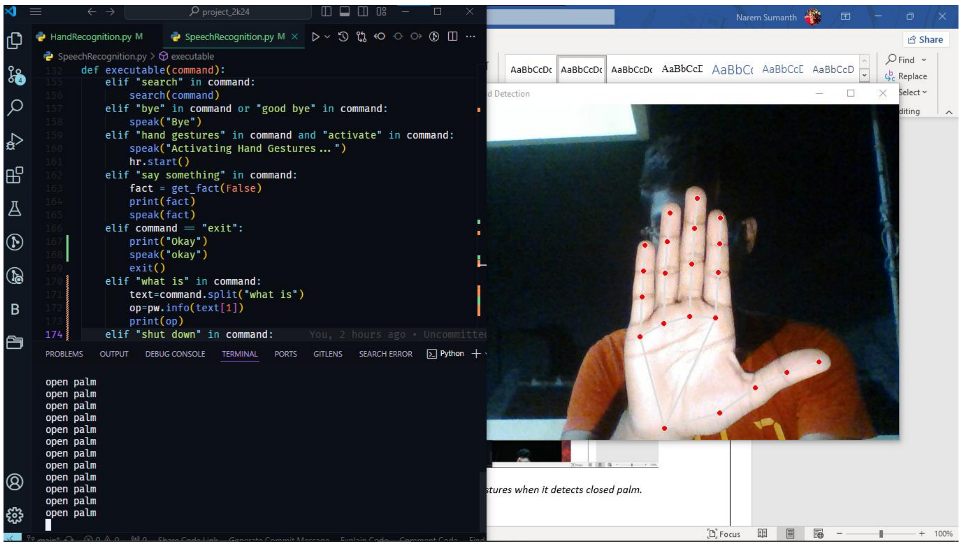
```
def y_t_date(c
    elif "yesterday's" in command or "yesterday" in command:
        print(obj.y_date)
        speak("yesterday's date is "+str(obj.y_date))
    elif "tomorrow's" in command or "tomorrow" in command:
        print(obj.t_date)
        speak("tomorrow's date is "+str(obj.t_date))
    elif "date" in command:
        print(obj.date)
        speak("today's date is "+str(obj.date))

def executable(command):
    num=[2,1]
    if "your name" in command or "who are you" in command:
        speak("My name is Elite, i'm your voice assistant")
        print("My name is The Elite, i'm your voice assistent
    if "tell me about" in command or "describe" in command ar
        print("I am Elite, a voice assistant developed in 202
        speak("I am Elite, a voice assistant developed in 202
```

```
open palm
open palm
open palm
open palm
open palm
open palm
open palm
open palm
open palm
open palm
118
closed palm
```

Fig 6.2.7: Deactivating Hand Gestures when it detects closed palm.

8. Hand Detection:

The system accurately detects hand gestures from a distance of 30 to 40 cm, ensuring reliable interaction and usability for users.



```
def executable(command):
    elif "search" in command:
        search(command)
    elif "bye" in command or "good bye" in command:
        speak("Bye")
    elif "hand gestures" in command and "activate" in command:
        speak("Activating Hand Gestures ... ")
        hr.start()
    elif "say something" in command:
        fact = get_fact(False)
        print(fact)
        speak(fact)
    elif command == "exit":
        print("Okay")
        speak("okay")
        exit()
    elif "what is" in command:
        text=command.split("what is")
        op=pw.info(text[1])
        print(op)
    elif "shut down" in command:          You, 2 hours ago • Uncommitted
```

```
open palm
open palm
open palm
open palm
open palm
open palm
open palm
open palm
open palm
open palm
open palm
open palm
```

Fig 6.2.: Detecting Hand Gestures successfully.

Test Case ID	Test Case Description	Expected Output	Actual Output	Result
TC1	Hand Gesture Detection from 30cm to 40cm away	Successful	Successful	**Pass**
TC2	Single hand detection even in multiple hands	Successful	Successful	**Pass**
TC3	Cannot work without internet	It won’t work without internet	It won’t work without internet	**Pass**
TC4	Activate hand gestures with keyword "activate hand gestures"	Activated Hand Gestures	Activated Hand Gestures	**Pass**
TC5	Open an application using the command "open <application>"	Application opens	Application opens	**Pass**
TC6	Understand multiple accents	Successful	Yes, It can Understand multiple accents	**Pass**
TC7	Play songs on YouTube upon user command	Song plays	It is playing song	**Pass**
TC8	Retrieve date and time accurately	Current date and time	Successful	**Pass**
TC9	Search for anything in a web browser	Relevant search results	Search content accurately	**Pass**
TC10	Run accurately with more than 90% accuracy	Accurate execution	Accurately understanding and generating outcomes	**Pass**
TC11	System compatibility test (System specs: i3 5th gen, 4GB RAM, 128GB SSD/HDD)	Can work without interruption	Can work without interruption	**Pass**

7. CONCLUSION

The fusion of hand gesture recognition and voice assistance brings a transformative era in human-computer interaction, promising a paradigm shift towards more intuitive and seamless control over systems and devices. This convergence of technologies not only revolutionizes the way we interact with machines but also opens up a plethora of possibilities for enhancing user experience and accessibility across diverse domains.

At its core, this fusion represents the marriage of two distinct yet complementary modes of human communication: visual and auditory. Hand gestures, a fundamental aspect of human expression, convey rich and nuanced information through movement and posture. Meanwhile, voice commands offer a natural and efficient means of communication, leveraging the power of spoken language to articulate intentions and commands. By combining these modalities, the proposed system transcends the limitations of traditional input methods, offering users a more intuitive and immersive interaction experience.

One of the key advantages of this fusion is its ability to cater to a wide range of users, including those with varying levels of technological proficiency and physical capabilities. For individuals with disabilities or impairments that may hinder traditional input methods such as keyboard and mouse interactions, hand gesture recognition and voice assistance provide alternative means of engaging with technology. By leveraging computer vision algorithms to interpret hand movements and gestures, the system empowers users to navigate interfaces and execute commands using natural gestures, regardless of their physical limitations.

Furthermore, the integration of speech recognition technologies enables users to interact with the system using voice commands, further enhancing accessibility and ease of use. This feature is particularly beneficial in scenarios where hands-free operation is desired, such as in automotive interfaces, smart home automation, or healthcare settings where sterile conditions prohibit physical contact with devices. By simply speaking commands aloud, users can initiate actions, retrieve information, and control devices with unprecedented convenience and efficiency.

Beyond accessibility, the fusion of hand gesture recognition and voice assistance holds immense potential for revolutionizing user experience across a myriad of applications and industries. In healthcare, for instance, the system can facilitate hands-free operation of medical equipment, allowing healthcare professionals to focus on patient care without being encumbered by cumbersome interfaces. In educational settings, it can foster interactive learning experiences by enabling students to engage with digital content through natural gestures and voice commands, promoting active participation and comprehension.

Moreover, in the realm of smart home automation, the system empowers users to control various appliances and systems with simple hand gestures coupled with voice commands. Whether dimming lights, adjusting thermostats, or playing music, the ability to interact with smart devices in a seamless and intuitive manner enhances the overall convenience and comfort of home environments.

In conclusion, the fusion of hand gesture recognition and voice assistance represents a watershed moment in human-computer interaction, redefining the way we engage with technology and unlocking new avenues for enhancing user experience and accessibility. By harnessing the synergies between computer vision and speech recognition technologies, the proposed system exemplifies the transformative potential of interdisciplinary approaches to innovation. As we continue to push the boundaries of technology, this fusion holds the promise of making interactions with machines more natural, intuitive, and empowering for users across diverse domains and demographics.

8. FUTURE ENHANCEMENTS

Future enhancements to the system hold the promise of further elevating the user experience and expanding the capabilities of the fusion of hand gesture recognition and voice assistance. By continually innovating and advancing the technology, we can unlock new levels of usability, accessibility, and functionality across various domains. One area ripe for improvement is the expansion of the gesture vocabulary and enhancement of recognition accuracy. While the current system may support a predefined set of gestures, future iterations can incorporate a broader range of gestures, allowing for more nuanced interactions. By expanding the vocabulary, users can perform a wider array of gestures to control devices, navigate interfaces, and execute commands with greater precision and specificity. Additionally, improvements in recognition accuracy will ensure that the system can reliably interpret and respond to user gestures, minimizing errors and enhancing overall usability.

Another key enhancement lies in the integration of global languages into the system. Currently, the system may be optimized for specific languages or dialects, limiting its accessibility to users from diverse linguistic backgrounds. By incorporating support for multiple languages, the system can cater to a broader user base, enabling individuals from different regions and cultures to interact with technology in their native languages. This not only enhances inclusivity but also improves usability and user engagement, as users can communicate more naturally and effectively with the system.

Enhanced security features for voice command authentication represent another area of future development. As voice commands become increasingly integral to interacting with technology, ensuring the security and privacy of user data becomes paramount. Future enhancements may include robust authentication mechanisms, such as voice biometrics or multi-factor authentication, to verify the identity of users and prevent unauthorized access. Additionally, encryption and secure communication protocols can be implemented to safeguard sensitive information transmitted during voice interactions, providing users with peace of mind and confidence in the security of their data.

Furthermore, future iterations of the system can explore compatibility with IoT (Internet of Things) devices to enable broader home automation functionalities. By seamlessly integrating with IoT ecosystems, the system can control a wide range of smart devices and appliances, offering users comprehensive control over their connected environments. Whether it's adjusting smart thermostats, monitoring security cameras, or managing smart appliances, the ability to interact with IoT devices through intuitive hand gestures and voice commands enhances convenience, efficiency, and comfort in the smart home environment.

In summary, future enhancements to the fusion of hand gesture recognition and voice assistance hold the potential to significantly enhance usability, accessibility, and functionality across diverse applications and industries. By expanding gesture vocabulary, integrating global languages, enhancing security features, and enabling compatibility with IoT devices, we can continue to push the boundaries of innovation and revolutionize human-computer interaction paradigms. As technology continues to evolve, these enhancements will play a crucial role in shaping the future of user experience and ushering in a new era of seamless and intuitive interaction with technology.

9. REFERENCES

The references provided serve as invaluable resources for gaining deeper insights into the technologies utilized in the project:

1. OpenCV Documentation: OpenCV, an open-source computer vision library, offers extensive documentation covering its APIs, functionalities, and usage. From basic image processing techniques to advanced computer vision algorithms. [https://docs.opencv.org/].

2. Media Pipe Documentation: Media Pipe, developed by Google, provides a suite of machine learning-based solutions for various multimedia tasks, including hand tracking and gesture recognition. Its documentation offers detailed explanations of its APIs, models, and sample applications, enabling developers to harness the power of Media Pipe for real-world projects. [https://google.github.io/mediapipe/].

3. Speech Recognition Library Documentation: The Speech Recognition library, available on GitHub, facilitates speech recognition in Python, supporting multiple APIs and languages. Its documentation elucidates the usage of different recognition engines, configuration options, and integration with other libraries, empowering developers to implement speech recognition functionality seamlessly. [https://github.com/Uberi/speech_recognition].

4. Speech Recognition PyPI Website: The SpeechRecognition library is also available on the Python Package Index (PyPI), providing easy installation and access for Python developers. The PyPI page offers additional information, such as version history, dependencies, and user reviews, making it a convenient resource for discovering and installing the library. [https://pypi.org/project/SpeechRecognition/].

5. Software Engineering:

Book Name: A Practitioner's Approach by Roger S. Pressman

Author: Roger S. Pressman

Soft copy:

amazon.com/Software-Engineering-Practitioners-Roger-Pressman/dp/0078022126

These external links serve as gateways to in-depth documentation, tutorials, and community support, enabling developers to leverage the full potential of OpenCV, Media Pipe, and Speech Recognition libraries in their projects.